FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Valtteri Inkiläinen

CLUSTERING IMAGE SETS WITH FEATURES FROM DEEP CONVOLUTIONAL NEURAL NETWORKS

Master's Thesis
Degree Programme in Computer Science and Engineering
September 2019

Inkiläinen V. (2019) Clustering image sets with features from deep convolutional neural networks. University of Oulu, Degree Programme in Computer Science and Engineering, 91 p.

ABSTRACT

This thesis compares the results of clustering image sets by features extracted using different layers of a convolutional neural network. The image features were extracted with layers of a pre-trained image classification network which layer weights were trained with ImageNet dataset. Eight image sets were used to test which extracted features achieve the best clustering accuracies. Features from the test image sets were extracted with the layers of the network architecture, and the features were clustered on a layer by layer basis. The clustering accuracies were measured with normalized mutual information (NMI).

The results show that the clustering accuracies depend on the characteristic of the image set being clustered. The image sets with more than two image categories had the best NMI scores with the features from the second last layer in the architecture, while the image sets with two categories had different layers give the best NMI scores. Moreover, the image set with blurred images had the best result come from few of the first layers, showing that the current method of selecting the second last layer for feature extraction in pre-trained CNNs is not always optimal.

Keywords: Feature extraction, transfer learning, dimension reduction, Xception, agglomerative clustering, UMAP

Inkiläinen V. (2019) Piirteiden vaikutus kuvaryhmän klusterointiin käyttäen konvoluutioverkolla irroitettuja piirteitä. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 91 s.

TIIVISTELMÄ

Tässä työssä vertaillaan kuvajoukkojen klusterointituloksia eri piirteillä. Piirteiden irrotukseen kuvista käytettiin valmiiksi koulutetun konvoluutio neuroverkon eri tasoja. Neuroverkko oli koulutettu kuvaluokitteluun ImageNet datajoukolla. Kahdeksan kuvajoukkoa klusteroitiin eri piirteillä, jotka oli irrotettu neuroverkon eri tasoilla. Näiden kuvajoukkojen klusterointitarkkuus mitattiin parhaan piirreirrotus tason löytämiseksi kullekin kuvajoukolle. Klusteroinnin tulos mitattiin normalisoidulla yhteisen informaation metriikalla (normalized mutual information).

Työn tulos osoitti, että klusterointitulos taso tasolta mitatessa riippuu klusteroitavasta kuvajoukosta. Kuvajoukot, jotka sisälsivät kuvia useammasta kuin kahdesta kategoriasta, klusteroituvat parhaiten verkon toiseksi viimeisellä tasolla irrotetuilla piirteillä. Kahden kategorian kuvajoukkojen parhaat klusterointi tulokset tulivat eri tasoilla. Kuvajoukko joka sisälsi kuvia sumeista ja tarkoista kuvista, saavutti parhaat klusterointitulokset piirteillä, jotka oli irrotettu verkon ylemmiltä tasoilta. Tulokset osoittavat, että yleisesti käytetty menetelmä valita valmiiksi koulutetun verkon toiseksi viimeinen taso piirreirrotukseen ei aina anna optimaalista tulosta.

Avainsanat: Piirreirrotus, siirto-oppiminen, dimensionaalisuuden pienentäminen, Xception, agglomeratiivinen klusterointi, UMAP

TABLE OF CONTENTS

FOREWORD

The advent of deep convolutional neural networks has sparked our curiosity for advancing machine learning. As new network architectures are developed at an ever-increasing speed, this thesis was done to inspect what capabilities lie inside of these networks and how these capabilities can be used in the domain of unsupervised learning. I had the privilege to research several machine learning topics for this thesis and deepen my knowledge of machine learning methodologies, which will have a great influence on my upcoming endeavours as a machine learning professional. This thesis would not have been possible without the support of Visidon Oy. I would like to thank the whole staff of Visidon for providing inspiration, ideas and great 8-pool matches with me during my thesis work. Special thanks to my technical supervisor Sami Varjo and principal supervisor Miguel Bordallo Lopez for their valuable feedback.

Oulu, 4th October, 2019

Valtteri Inkiläinen

ABBREVIATIONS

AC	Agglomerative Clustering
ANN	Artificial neural network
ARI	Adjusted Rand index
CNN	Convolutional neural network
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
HDBSCAN	Hierarchical DBSCAN
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
NMI	Normalized Mutual information
PCA	Principal component analysis
ReLU	Rectified linear unit
SGD	Stochastic gradient descent
UMAP	Uniform Manifold Approximation and Projection

K	number of clusters
ε	minimum distance
$minpts$	minimum number of points
$d(x_p, x_q)$	pairwise distance
d_{core}	core distance
d_{mreach}	mutual reachability distance
$G_{m_{pts}}$	Mutual Reachability Graph
λ	density level
E_R	*Relative Excess of Mass*
$S(C_i)$	cluster stability
σ	standard deviation

1. INTRODUCTION

The goal of image set clustering is to group similar images to the same groups and dissimilar images to different groups. Let us have an example of an image set clustering. There is probably hundreds if not thousands of photos in a smartphone. If I would ask to group all of the photos to subgroups, with similar images in the same group and dissimilar images in different groups, what kind of subgroups there would be? Maybe a group of photos including pets, or a group of selfies taken on vacation with family, or even a group of failed photos. The boundary of similar and dissimilar photos in groups is intrinsically related to the set of photos you have.

To measure the image similarity algorithmically, we need to find a measure from the images which can be used to compute the similarity of the images. For example, one could use similarity measurement between the pet photos and vacation selfies. In the domain of computer vision, images are commonly reduced to features for classifying algorithms to be able to classify the complex concepts of images [1]. The state-of-the-art image featurization methods are a group of machine learning techniques called artificial neural networks (ANNs), and precisely the subgroup of deep convolutional neural networks (CNNs). Convolutional neural networks are non-linear estimators that can be used, for example, to classify images to categories by training the network to learn features from the images. The network is trained with example input-output pairs of images and their corresponding category, e.g. a group of vacation selfies which are categorized as selfie and group of pet images which are categorized as pets.

CNNs consist of different layers of mathematical operations which are stacked on top of another, previous layers output being the input of the next layer. Each of these layers learns to extract particular features known as feature-maps. The nature of how CNNs are trained means we do not know exactly what features each layer learns to extract, but the principle is that first layers learn more general features, i.e. features found in many images, like patterns of simple shapes, while deeper layers learn more complex features i.e. features which are more specific to the images in the training image set [2].

Now in this thesis, we were interested in using the different features which can be extracted with pre-trained CNN layers, as the clustering similarity measurements to test how different image sets can be clustered with these features. Relative works similar to this thesis are [2, 3], where [2] shows how the features change from general features to more specific features when deeper layers of pre-trained CNN are used to extract the image features. They also discussed on the transferability of the features to other tasks i.e. using the pre-trained CNN for classification on other image sets. In [3], the researchers showed how image features extracted with pre-trained CNN can be used to cluster images to categories. They also briefly mentioned how selecting different layers to extract the features affect the clustering accuracy. This thesis studies more closely how the selection of different layers as feature extractors affect the clustering accuracies of the image sets by testing multiple layers as feature extractors.

The benefits for testing the feature extraction capabilities of pre-trained network is in the resources needed to train new networks from scratch [4, 5]. If the

existing network's layers can be used for other feature extraction tasks, it shows a clear benefit as there is no need to train a new network.

The experimental tests were done in a premise that if the image sets could be clustered to correct groups, the layer had extracted features from the images which distinguish the images from another, and thus these features could be used as a similarity measurement between the images in the selected image set. As noted in [2], deeper layers learn to extract features more specific to the training set of the CNN, and upper layers learn to extract more general features. Thus testing all of the layers of one CNN architecture was conducted to find out which kind of similarities could be measured from the extracted features.

The scope of this thesis is limited to one CNN architecture, the Xception architecture developed in Google [6], with pre-trained weights for image classifying, trained with ImageNet dataset. Following the results of [3], which shows that image set clustering paired with Xception as feature extractor gives the best results in image clustering tasks.

The implementation part of the thesis compares how the features extracted with each layer are clustered. The cluster accuracies are measured by using labelled image sets and external validation metrics. The features are extracted from multiple image sets, and clustering accuracies are measured for each. The selection of the clustering algorithm and dimension reduction method for the clustering is also considered.

This thesis contains first the introduction to cluster analysis and examples of the used clustering algorithms. This is followed by an introduction to convolutional neural networks and the used Xception architecture. Dimension reduction of the features for clustering is briefly discussed in chapter four. The used datasets are presented in chapter five. The implementation and results are displayed in chapter six. Lastly, the discussion of the results is held in chapter seven, followed by conclusions in chapter eight.

2. CLUSTER ANALYSIS

Backer and Jain define cluster analysis as "*In cluster analysis, a group of objects is split up into a number of more or less homogeneous subgroups on the basis of an often subjectively chosen measure of similarity, such that the similarity between objects within a subgroup is larger than the similarity between objects belonging to different subgroups.*"[7], while Hansen and Jaumard write "*Given a set of entities, Cluster Analysis aims at finding subsets, called clusters, which are homogeneous and/or well separated.*"[8]. These definitions give the basis of clustering, which is to find a subset of a given set in which each subset contains similar objects and other subsets having dissimilar objects. The similarity measurement of images in the image set could be the shape of an object in the image, or the colour of the object, or some other measurable quantity from the image. In this thesis, the set of entities or group of objects which are being clustered are the features extracted from the images. The clustering algorithm's objective is to find separable subgroups of these features.

The clustering techniques explained in this chapter are categorized as partitional clustering, hierarchical clustering, and density-based clustering algorithms, based on the properties of the generated clusters. Partitional clustering algorithms directly partition the group of objects to a predefined number of clusters, without defining any hierarchy for the clusters [9 p.63]. Hierarchical clustering algorithms construct a hierarchy of clusters by partitioning the data sequentially, generating hierarchy from where all of the data points are in one cluster to each data point being a single cluster or vice versa [9 p.31]. Density-based clustering algorithms cluster data points by representing the data points by their density conditions (distance and number of nearest neighbours) on the clustering space [10]. Density-based clustering algorithms are tolerant to noise, as low-density points can be disregarded from the clustering as outliers or noise [11].

This chapter introduces the most common clustering methods, including the methods used in the implementation part of this thesis, the agglomerative clustering and the current state-of-the-art HDBSCAN. There is also a brief discussion of how the clustering parameters are selected for the implementation part, and the chapter finishes on summarising different validation metrics for clustering.

2.1. K-means

K-means clustering is a partitional clustering algorithm [9 p.68]. K-means partitions the dataset to K groups in an iterative process. In native K-means implementation, the number of wanted clusters is given to the algorithm as the K parameter. The algorithm is initialized by assigning a K number of cluster centres (centroids) at random on the dataset space, initially partitioning the dataspace to K partitions. The partitioning is updated in alternating between two steps. The two steps consist of assigning the data points to the nearest centroid, by calculating the Euclidean distances of the data points to the centroids, and updating the centroid locations based on the calculated mean values of the data points assigned to the centroid. The algorithm finishes when no changes appear

to the clusters, e.g. the clusters remain stable. [9 p.68]. The K-means algorithm steps are illustrated in Figure 1.

The advantage of K-means is that it is a fast and quite simple algorithm to implement. It also clusters all the data points, i.e. it does not have a noise group. This can also be seen as a limitation of the K-means because it needs to expand the clusters to outliers which can alter other quite densely packed groups. Other limitations of the K-means are the random initialization of the algorithm, which means the partition changes between runs, and that the K parameter needs to be given before the clustering, i.e. the optimal number of clusters is left to be determined by the researcher.

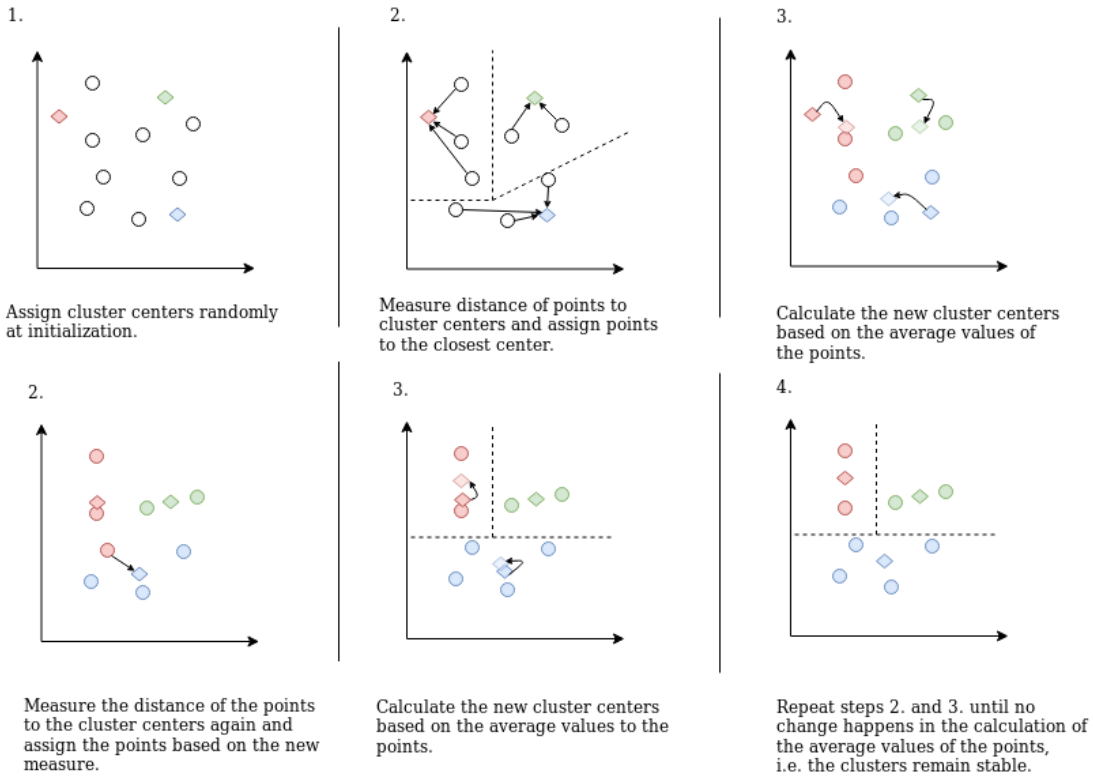

Figure 1. Illustration of the K-means algorithm. In this example, the K-parameter is set to 3.

2.2. Agglomerative Clustering

Agglomerative Clustering or (AC) is a hierarchical clustering method. In AC each data point being clustered is first considered being on their own cluster [9 p.32]. These clusters are merged by their proximity (distance function) to another cluster. This merging is continued until all the points are assigned to one cluster, or when there is a wanted number of clusters.

The parameters for the AC are the distance function and the number of wanted clusters. The distance function is used to measure the proximity of the clusters to be merged, which is also called the linkage of the clusters [9 p.33]. One link-

age method is Ward's method, which minimizes the change in variance between clusters, and merges the clusters with the minimum value of this criterion [12]. AC generates hierarchy for the merged clusters and the wanted number of clusters is selected by cutting the hierarchy on the level which results for the wanted number of clusters. Agglomerative clustering can be visualized as a tree structure, where single clusters are leaf nodes and merged clusters branch downwards the root, this is illustrated in Figure 2.

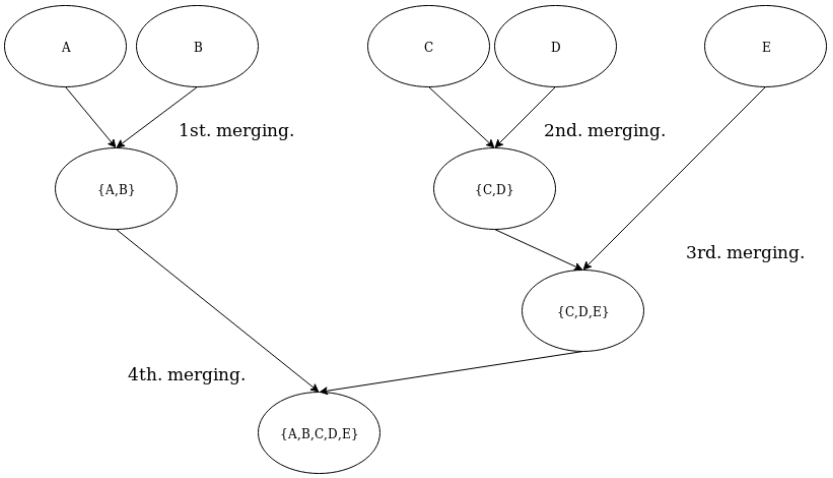

Figure 2. Agglomerative clustering visualization as tree structure. (A), (B), (C), (D) and (E) points are first their own clusters. The closest clusters of each other by the selected distance metric are the clusters (A) and (B), which are merged to cluster {A, B}. In the second step the distance of clusters is measured between clusters {A, B}, (C), (D) and (E), the clusters closest to each other in the second step are the clusters (C) and (D). This measuring of closeness and merging is continued and in the third step the closest clusters are the cluster {C, D} and (E) and in the fourth step the clusters {A, B} and {C, D, E} are merged.

2.3. DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is a density-based clustering algorithm developed by Ester *et al.* [13]. DBSCAN operates by grouping data points which are packed densely together to clusters and points which are far away from other points (low-density regions) to noise. The points which are being clustered can have three properties in the DBSCAN point of view, the point can be a core point, a reachable point, or an outlier (noise). The parameters needed for DBSCAN are the minimum distance or radius (marked as epsilon ε) and minimum points (*minpts*) parameters. ε is the metric from which distance away from the other points, a point is determined to be reachable. The *minpts* parameter sets how many points need to be reachable of the point (the point itself included) to be converted as a core point. The core points form clusters with the reachable points that are within the given ε

distance from the core point. Outliers are points which are non-reachable to the core points. [9 p.221]

Clusters are formed with all the points reachable by the core point (including, other core points). New clusters are formed when core points are not reachable from each other. Different steps of the algorithm are illustrated in Figure 3.

The advantage of DBSCAN is the ability to generate clusters with arbitrary shapes and good scalability [9 p.220]. Some of the limitations of the DBSCAN is the choice of suitable parameters [14]. The *minpts* parameter can effectively restrict the forming of clusters. If the minimum points needed to form core points is set too great, natural clusters with fewer data points might be clustered as noise. The selection of the distance parameter ε can be problematic for high-density datasets. Selecting too high ε will cause all the data points to become reachable and then be clustered to one cluster. Selecting too small ε in already densely packed points might cause a division into multiple clusters. As the parameters are the same for the whole dataset being clustered, DBSCAN can only provide clusters based on a global density threshold [14].

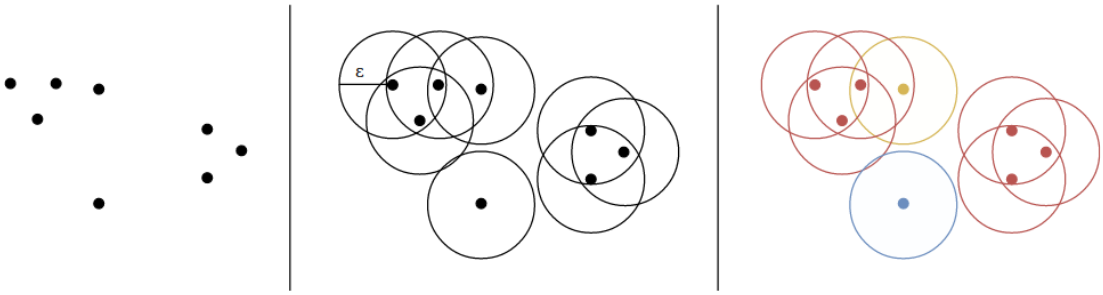

Figure 3. Steps of DBSCAN algorithm. The minimum distance is set to epsilon ε and the minimum points is set to 3. The first column shows the points. In the second column, the distance ε is drawn for all the points. The last column shows the assigning of the points to the core, reachable and noise points. The red points are core points because the red points are within ε distance from the given *minpts* (3 points, with the point itself included). The yellow point is reachable point because it is within ε distance of core point (red). The blue point is an outlier (noise) because it is not within the distance ε from the core point. The left side red points plus the yellow point are one cluster, while the red points in the right side are considered a second cluster and the blue point is clustered to noise group.

2.4. HDBSCAN

Hierarchical density-based spatial clustering of applications with noise (HDBSCAN) is a density-based clustering algorithm, based on hierarchical density estimates. HDBSCAN was derived from DBSCAN by Campello *et al.* [14] and extends DBSCAN to a form of hierarchical clustering algorithm. HDBSCAN performs DBSCAN over differing epsilon (ε) values and tests the result to find a clustering that grants the best stability over epsilon [15].

HDBSCAN removes the need for giving the minimum distance parameter epsilon ε in DBSCAN, by running the DBSCAN only with minimum points ($minpts$) parameter. This indicates that every point ($x_p \in X$) of the dataset will find the $minpts$ number of nearest neighbour points with differing pairwise distances ($d(x_p, x_q)$). The distance or radius which encloses the $minpts$ number of points is defined as core distance (d_{core}). Figure 4 illustrates the core and pairwise distances with given $minpts$ parameter. The core distance is needed for calculating the mutual reachability distance (d_{mreach}). Mutual reachability distance is calculated with Equation (1) [14]

$$d_{mreach}(x_p, x_q) = max\{d_{core}(x_p), d_{core}(x_q), d(x_p, x_q)\} \tag{1}$$

The mutual reachability distance transformation is used to generate a weighted graph, which in the original paper is called the Mutual Reachability Graph ($G_{m_{pts}}$) [14]. The points of the dataset ($x \in X$) are the graph vertices and the d_{mreach} is the weight of the edges between the points it was calculated. From this graph, a

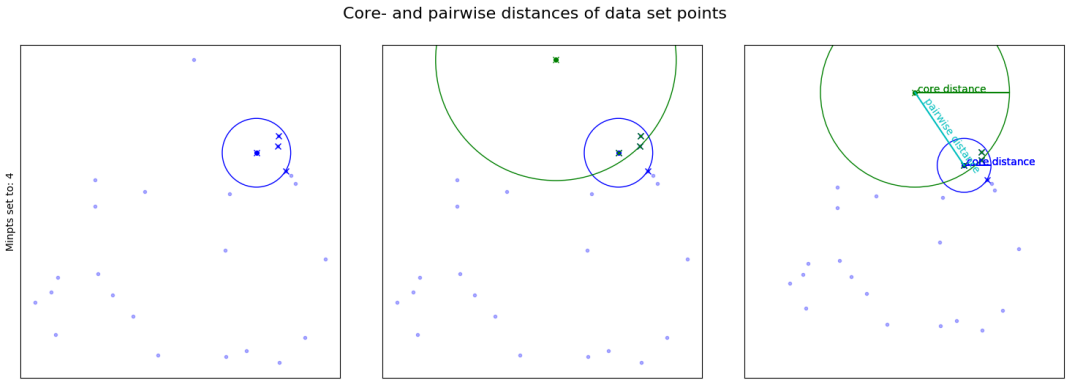

Figure 4. The points and their core distances with $minpts$ value set to 4. The pairwise distance is calculated between the points. Mutual reachability distance is calculated between points by the Equation (1), which dictates that the d_{mreach} should always be the higher value between the core distances and pairwise distance.

minimum spanning tree (MST) is extracted via the MST finding algorithm[1]. The resulting MST is shown in figure 5. The MST can be converted to a hierarchy of connected components by sorting the MST edges by their distance and iterating through the edges, and then merging the edges with the closest edge to new groups [15].

The result of the merging of the edges can be viewed as a dendrogram (a diagram representing a tree structure), which is shown in figure 6. From this dendrogram, we could get clusters similar to in DBSCAN by drawing a horizontal line through the dendrogram by the distance value of the parameter epsilon. However, in HDBSCAN this is done automatically by condensing the dendrogram to

[1]Graph theory and the related minimum spanning tree algorithm are out of scope of this thesis, one used minimum spanning tree algorithm is Prim-Djikstra algorithm which is defined in paper [16].

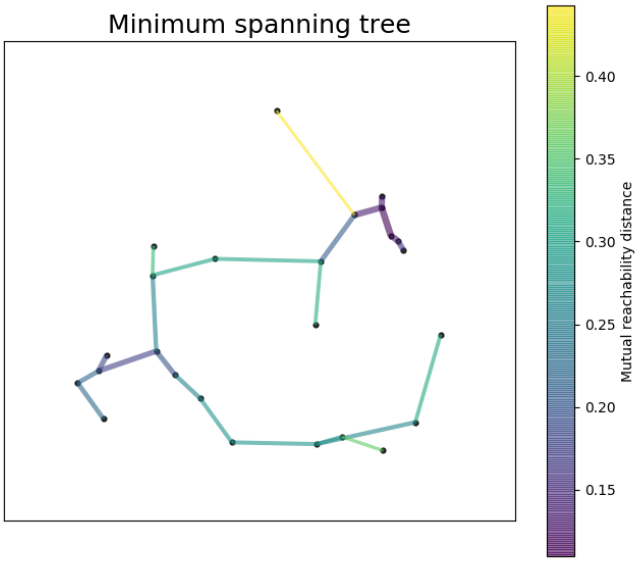

Figure 5. Minimum spanning tree generated from the $G_{m_{pts}}$. Figures 5-8 generated via HDBSCAN software package [15]

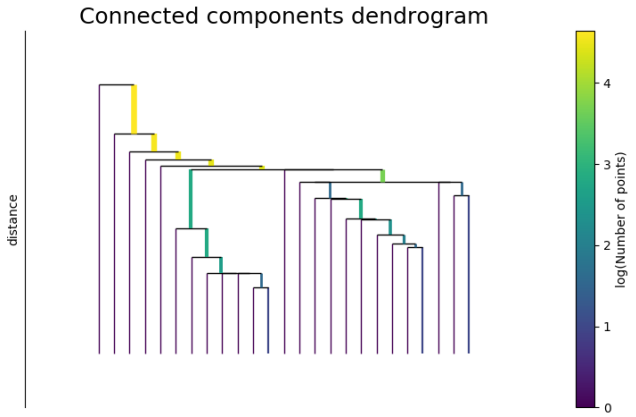

Figure 6. Connected components dendrogram, generated from the minimum spanning tree of the graph $G_{m_{pts}}$.

a less branched dendrogram based on the stability of the clusters. The condensed dendrogram is shown in figure 7. The stability of the cluster C_i at level λ in the dendrogram is calculated with the concept of *Relative Excess of Mass* (E_R) of a cluster C_i. The E_R at level $\lambda_{min}(C_i)$ is calculated by Equation (2) [14]

$$E_R(C_i) = \int_{x \epsilon C_i} (\lambda_{max}(x, C_i) - \lambda_{min}(C_i))dx \tag{2}$$

where the $\lambda_{max}(x, C_i)$ is the density level where C_i is split or it disappears. The discrete case of Equation (2) for finite dataset size X is formulated as Equation (3) [14],

$$S(C_i) = \sum_{x_j \epsilon C_i} (\lambda_{max}(x_j, C_i) - \lambda_{min}(C_i)) = \sum_{x_j \epsilon C_i} (\frac{1}{\varepsilon_{min}(x_j, C_i)} - \frac{1}{\varepsilon_{max}(C_i)}) \tag{3}$$

where $\lambda_{min}(C_i)$ is the minimum density level at which C_i is defined as cluster. The parameter $\lambda_{max}(x_j, C_i)$ is the maximum density level where the object x_j belongs to cluster C_i, and $\varepsilon_{max}(C_i)$ and $\varepsilon_{min}(x_j, C_i)$ shows the correlation of λ level to corresponding ε values [14]. Now the clusters can be selected from the condensed dendrogram (figure 8). The resulting selection shows that two clusters were found in the data and the data points outside of the selected λ values are grouped as noise. The clustered dataset is shown in figure 9, where the dark points belong to the noise group and the yellow and green points are the points that are assigned to the clusters.

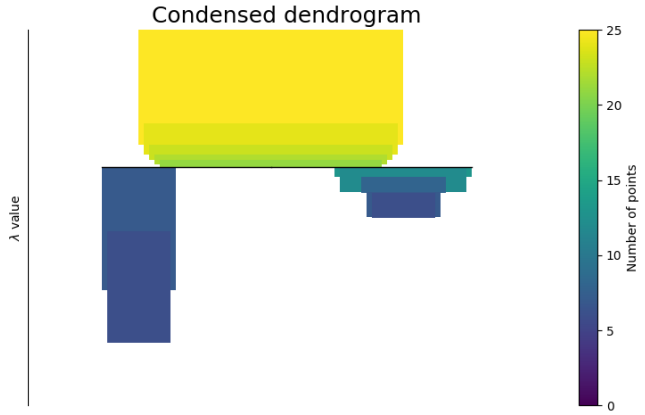

Figure 7. Condensed dendrogram, by the stability of the clusters.

HDBSCAN's advantage over DBSCAN is that it is not limited to one density threshold. HDBSCAN can calculate DBSCAN like densities with an infinite range of density thresholds and construct a simplified tree structure from the most significant clusters. From this tree structure, the optimal threshold can be selected based on the stability of the clusters [14], effectively removing the problem of selecting optimal ε in DBSCAN. Even the added steps in HDBSCAN does not make it slower than DBSCAN [17] and such is a generally a better alternative to use over DBSCAN.

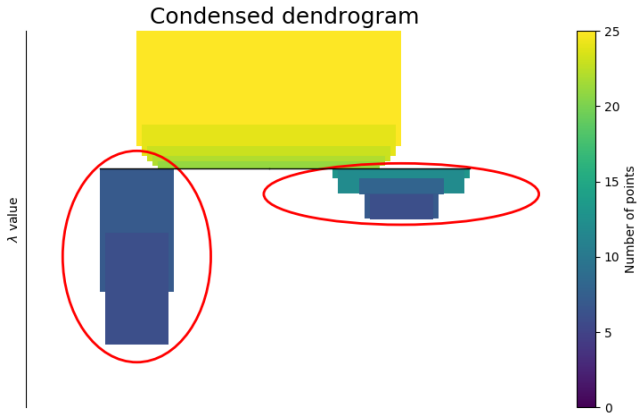

Figure 8. Select the clusters based on the stability $S(C_i)$.

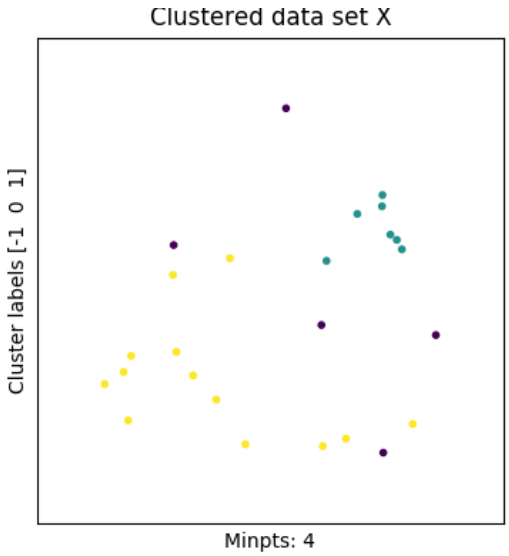

Figure 9. The resulting clustering of the data set X with HDBSCAN.

2.5. Clustering parameter selection

Many of the discussed clustering algorithms have parameters that need to be assigned before clustering. The main parameters for the algorithms used in the implementation are the number of wanted clusters in the case of AC and the minimum points in HDBSCAN. The other parameters are the distance metric to calculate the distances between the points in HDBSCAN, and the linkage method in AC. These parameters affected the clustering outcome, and usually, they are selected based on the dataset being clustered. However, this needs to be done automatically or left for default values if the dataset is unknown.

In this thesis, the number of wanted clusters for the AC algorithm is selected based on the dataset being clustered. The number of groups in the datasets are known as the images in the datasets have labels on them. The linkage method for AC is selected to be Ward's method.

The distance metric parameter is set to Euclidean, which is the default distance metric in the implementation of HDBSCAN. In HDBSCAN the number of minimum points is left for the default value of 5. This default minimum points value is not optimal for each of the datasets, but as leaving the value as default, we eliminate the parameter tuning for this clustering algorithm and only focus on how the under-laying features affect the clustering outcome.

2.6. Cluster validation

Validation of the clusters is derived based on a selected metric once the data set is clustered. The clustering validation metrics can be classified into three groups, which are internal criteria, external criteria and relative criteria [9, 10]. The internal validation criteria are used for unknown data sets without labels, and such, the validation of the clusters could be the clustering object itself. External validation criteria can be used in data sets with known structures, like image-sets with image labels. Relative validation compares two clusterings of the dataset, either with the same algorithm with different parameters or clusterings with different clustering algorithms. The next sections explain the validation criteria more in-depth and introduce the selected algorithms that are used for validating the cluster outcomes in this thesis.

2.6.1. Internal criteria

Internal criteria are used to measure the validity of the generated cluster structure without prior (external) information about the clusters. The internal validity comes from the objective of the clustering, which was generating groups where similar objects are in the same group and distinctive objects in different groups [18]. The basic measurements for validating unknown data sets are the compactness and separation of the data points in the generated clusters. The internal validation methods do not give insight about if the clusters are correct or mean-

ingful on the application domain, but the internal validation measurements can be used to calculate the optimal number of clusters for the given data set [18].

One algorithm used in measuring the quality of the cluster structure is the silhouette technique [19]. The silhouette value is calculated for one data point at a time, based on the distances to other clustered data points. The silhouette value scales in a range from -1 to 1, 1 meaning the data point is appropriately clustered and -1 means the data point would naturally belong to the neighbouring cluster, i.e. the cluster it belongs is "wrong" based on the distance calculated. The silhouette value for point i is calculated at the following Equation (4) from [19]:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{4}$$

where $a(i)$ is the average dissimilarity of point i to all other objects in the cluster it is assigned, marked as A. $d(i, C)$ is the average dissimilarity of i to all objects in the other clusters C. $b(i)$ is calculated as $b(i) = min_{C \neq A}(d(i, C))$ for all the clusters C, except for the cluster the i belongs (A). From Equation (4), we can formulate the range as $-1 \leq s(i) \leq 1$ [19].

2.6.2. External criteria

External criteria assume having information about the data structure before clustering the data. After clustering the data, we compare this prior information to the generated structure of the clustering [10]. In image-set clustering, this prior structure of the dataset means we have prior knowledge about which images belong to the same group and which don't, e.g. the labels of the images. Two example indexes for measuring the external validation are the Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI). The Rand index is a measure of similarity between two clusterings [20] and the Adjusted Rand Index, is adjusted for the chance of grouping elements to correct groups, i.e. ARI outputs similarity values for groups with more similarity than randomly assigning elements to clusters.

The ARI is calculated using the Equation (5), where U and V are different clustering of the same dataset, e.g. the U could be the clustering results of running the clustering algorithm, and the V could be the clustering based on the given labels for the images.

$$ARI(U, V) = \frac{2(N_{00}N_{11} - N_{01}N_{10})}{(N_{00} + N_{01})(N_{01} + N_{11}) + (N_{00} + N_{10})(N_{10} + N_{11})} \tag{5}$$

The indexes, N_{00}, N_{01}, N_{10} and N_{11} are defined as follows: N_{11} is the number of data point pairs that are in the same cluster in both U and V clusterings. N_{00} is the measurement of pairs in different clusters in both U and V, while N_{01} is the number of pairs which belong to the same cluster in U but are in different clusters in V. N_{10} is the number of pairs which belong to the same cluster in V but are in different clusters in U. [21] ARI scores near 0.0 means random labelling and ARI score of 1.0 means perfect match [22].

The NMI is information theoretic based measure and NMI is defined via marginal and joint distributions of data in the clusterings U and V. The mutual information (MI) is expressed as $I(U,V)$ which is defined as $H(U) - H(U|V) = I(U,V)$. $H(U)$ and $H(U|V)$ are calculated as in [21]

$$H(U) = -\sum_{i=1}^{R} \frac{a_i}{N} log \frac{a_i}{N},$$

$$H(U|V) = -\sum_{i=1}^{R}\sum_{j=1}^{C} \frac{n_{ij}}{N} log \frac{n_{ij}/N}{b_j/N}$$

The MI can be normalized with multitude of generalized mean methods like *joint*, *max*, *sum* and *sqrt*, just to name a few [21]. In this thesis, the MI is normalized with the *sqrt* method and the NMI is calculated as in Equation (6) [22]:

$$NMI(U,V) = \frac{I(U,V)}{mean(H(U), H(V))} \tag{6}$$

External criteria give a valid measurement of the clustering quality, but as it relies on the class labels, it is not a suitable measurement for datasets without labels or unknown data.

2.6.3. Relative criteria

Relative criteria can be seen as a measure of the utility of the clustering algorithm. Relative validation criteria can be the selection of clustering parameters or clustering algorithms, which best fit the dataset being clustered [10]. In this thesis, the relative validation criterion is to use two clustering algorithms for the selected datasets. In the labelled datasets, both clustering algorithms use NMI for external validation, and the resulting NMI values are used to evaluate if the change in the NMI values were due to differences in clustering algorithms or the differences from the datasets.

2.6.4. Other validation methods

Manual inspection can be used as a validation method when the objects being clustered can be inspected, like images. The manual inspection of clusters can be used to verify unlabelled datasets, where the usage of the external validation measures are not possible. Manual validation is usually conducted by a domain expert.

3. CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks (CNNs) are non-linear function estimators, which work by the principle of learning a function f which maps the given inputs x to some defined outputs y,

$$f(x) = y \tag{7}$$

by learning the function with example input-output pairs. In the case of image classification CNNs the network is trained on with input images and their corresponding classes. After training the CNN, it is used to predict on which class the feed input image belongs. The main difference of convolutional neural networks over other artificial neural networks is that they have least one or more convolution layers, which can be used to process data that has grid-like topology, like image data, which can be seen as a 2-D grid of pixels [23 p.326].

CNN derives its idea from biological visual systems where studies of the mammal visual system [24] determined that the first parts of the visual system respond to patterns of light coming from simple shapes to the receptive fields. This idea is carried over to CNNs as the convolution operation on the first layers of the architecture tries to mimic this biological process by activating on similar shapes of the input image as detected by the biological receptive field of cells. The first implementations derived from this idea date back to 1980 when Fukushima presented the Neocognitron [25].

One of the first modern CNN for visual recognition was the network LeNet-5 which was trained using backpropagation and was used for handwritten digit recognition [26]. More recent CNNs use similar stacked-layer architectures which use tens to hundreds of layers, while their predecessor LeNet-5 contained only seven layers. Modern architectures use a multitude of different techniques to increase the accuracy of the networks and ease their training. On the next section, the basic structure of a CNN and the elements within are explained.

3.1. Structure

The CNN structure depends mostly on the selected architecture, but they all share similar basic building blocks. The building blocks of any CNN are the layers of the CNN. Usually, the layers of CNNs consist of operations like convolution, activation, and pooling. These operations are illustrated in figure 10. The convolution operation is the dot product of the input and the layer's kernel (or filter). This produces a set of linear activations. The activation function is used to scale the linear convolution outputs to non-linear activation maps. Pooling operation modifies the input by taking, for example, average or maximum values of certain sized rectangle neighbourhood of the input and passing these values to the next layer effectively downsampling the input.

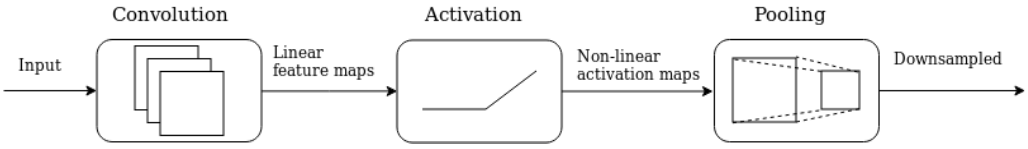

Figure 10. The basic operations of convolutional neural network.

A dense layer with softmax activations is operation found in the last layer of the network. The dense (or fully connected) layer with softmax activation functions is used to map the inputs to the predicted class. Softmax function works by distributing the input values to the layer as predictions from 0 to 1 to the classes, and the highest prediction being the class the network has classified the input image. Figure 11 shows an example of image classification done with CNN.

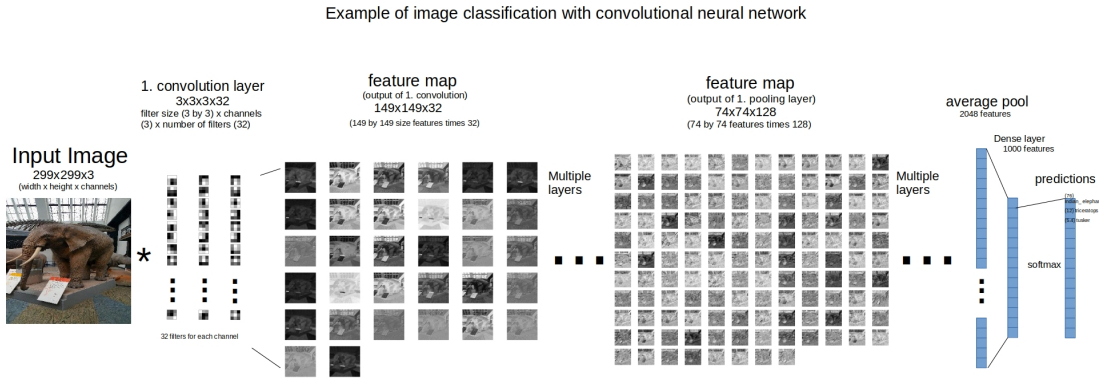

Figure 11. Image classification example with Xception CNN architecture. The CNN takes RGB images sized 299 by 299 pixels as input. The first convolution layer has 32 filters for each channel (RGB) and each filter consist of 3×3 size kernel, with trained weights. The output of the first convolution layer is the convolution between the filter weights and the input images channels, each channel result summed as feature map with 32 features size 149 by 149. The previous layers feature map is feed to the next layer, which results for another feature map. In the average pooling layer, the feature map is reduced to a vector of size 1×2048, which is feed to a dense layer, which reduces the features to a vector of size 1×1000. From this feature vector, prediction of the image class is concluded by taking the softmax of the vector and mapping the highest value to the corresponding class.

3.1.1. Convolution layer

The heart of the CNNs is the convolutional layer. The convolutional layers consist of a group of kernels which have learnable weights (or parameters). The convolution operation is a dot product over vectors, which is presented in Equation (8) (from [23 p.328, Fig.9.4]). The equation is a discrete case of the convolution shown for two-dimensional image (I) and kernel (K).

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{8}$$

When the convolution is expanded to colour images, each colour channel is convolved separately by the corresponding kernel, which means that three channel colour image needs three separate kernels or a 3-dimensional filter. An example of a 3-dimensional convolution operation is illustrated in Figure 12. The dimensions of the feature map (output of the convolution) in the example are 2×2×1 because the convolution is summed over all the channels, e.g. the selected 3×3×3 filter has 27 values and the dot product is taken by these values and the corresponding area of size 3×3×3 of the image. As the image is larger in x and y dimensions, the filter is slided over the image, such that the whole image is convolved sequentially.

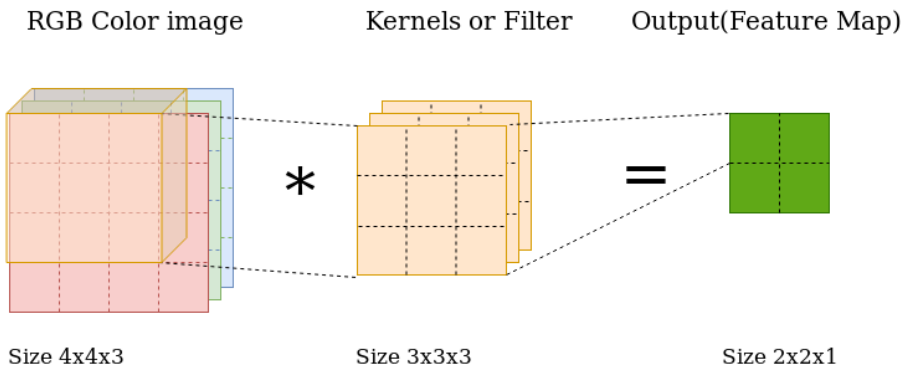

Figure 12. Example of a 3-dimensional convolution for three channel colour image.

While the kernel size may vary depending on the implementation, the kernel size is usually smaller than the input size. When the kernel size is smaller than the input the convolution is said to have sparse interactions or sparse weights. This property is important to CNNs because images can have thousands or millions of pixels, but even small-sized chunks of the image can be used to extract important features of the image like edges. With this property, the filters can extract meaningful features from the image, with fewer parameters than having to save the parameters for every pixel. [23 p.330]

The input image's size reduction can be controlled by zero-padding the input image's edges with a layer of zeros. For the example in figure 12, zero-padding the image would result in the image's size to be raised to $5 \times 5 \times 3$ and the convolution with the same kernel size of $3 \times 3 \times 3$ would result in output of size $3 \times 3 \times 1$. The reduction of input image's size can be calculated with Equation (9) where O is the output size of height h and width w, while I is the image size and f is the kernel size.

$$O(h,w) = (I_h - f_h + 1), (I_w - f_w + 1) \tag{9}$$

The result of the convolution (feature map) is feed to the next operations of the architecture. The convolution operation works the same for feature maps. The

parameters of the convolution layer, are the weights of the kernel, the dimension of the kernel and stride which are explained in section 3.1.5.

3.1.2. Activation layer

Activation layer's primary function is to add non-linearity to the network. Non-linearity is needed to approximate non-linear problems. There is different non-linear activation functions which can be used like *sigmoid*, *tanh*, ReLU and their variants. ReLu is probably the most commonly used activation function in CNNs. ReLU maps the output values of the convolution to values from zero to upwards via transfer function as shown in Equation (10) [27].

$$f(x) = max(0, x) \tag{10}$$

This way the negative values are removed from the convolution output and positive values have linear mapping, making the ReLU non-linear with the transformation. The activation transformation results in an output which is referred to as the activation map.

3.1.3. Pooling layer

Pooling layers act as statistical summaries of the feed input by calculating a single value from the neighbouring values of the input. The benefits of pooling are to make the input spatially invariant to small translations, which correlates for better generalisability of the features, and downsampling the input, making the following computations more efficient. For example in the case of max-pooling, the max value of the selected input area is passed to the next layer. If this max value's position changes a little, but within the area of the pooling operation the same max value is selected, and the pooling layers output stays the same [23 p.335-338]. However, the downsampling in max-pooling causes aliasing[1] which can decrease the accuracy of the network [29]. The suggested solution for the aliasing problem is to add anti-aliasing filter to the pooling operation.

The pooling layer parameters are the pooling operation, the size of the pooling operation, and the stride. Two common pooling operations are max-pooling, which selects the maximum value in the pooling area, and average pooling, which averages the values of the pooling area and passes this value forwards. Max-pooling is illustrated in Figure 13.

[1]Aliasing is a phenomenon related to the sampling theorem. Aliasing occurs when the sampling of signal (discrete in digital) is below the Nyquist frequency. More of the sampling theorem, effects of aliasing and how to mitigate them with anti-aliasing can be found in the book 'Practical Digital Signal Processing' by Edmund Lai [28 p.15]

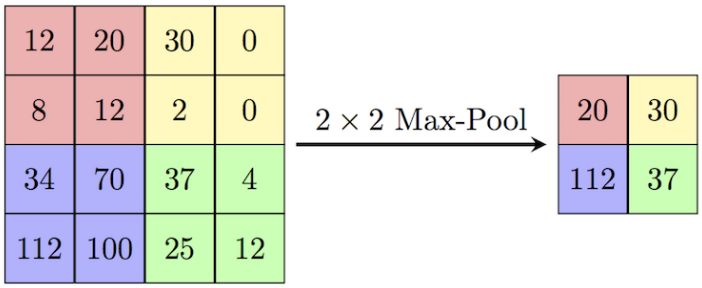

Figure 13. Max-pooling with pooling area size 2×2 pixels and stride 2. Figure from [30].

3.1.4. Other elements of advanced CNNs

As the available computing power increases, more complex architectures have been developed. Usually, the simplest way to achieve improved classification accuracies is to increase the labelled training data, but as the training data increases, so does the training time. Better architecture designs are constantly needed. Some additions to the typical set of layers and techniques are residual learning [31], batch normalization [32], inception module [33] and depthwise separable convolutions [34], which further increased the accuracy and trainability of the networks.

Residual learning adds skip connections to the network, where layer outputs are added to inputs of layers deeper in the architecture, as well as passing the output to the next layer [31]. The network has fewer layers to learn in the early stages of training, because of the skip connections, and thus easing the problem of vanishing gradients (for vanishing gradient, see section 3.2). A residual block is illustrated in figure 14.

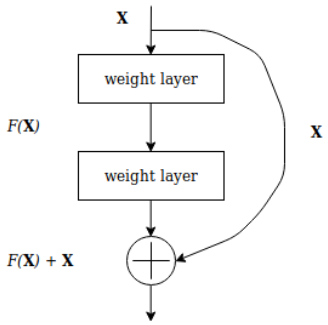

Figure 14. Residual learning, figure modified from [31].

Networks that use batch normalization can be trained with higher learning rate (for learning rate, see section 3.2), which theoretically accelerates the training of the network. Ioffe and Szegedy argue that the batch normalization works by reducing the internal covariate shift [32], but some experimental studies [35] show that this might not be the cause. The exact mechanism of how the batch normalization works is still discussed, but the benefits of using it are evident [36].

A Inception module adds multiple convolutions with different kernel sizes to one layer, making it wider. The idea behind making the network wider is that the kernel size limits how small or big features the kernel can search from the image. When adding three convolution kernels with varying size (1×1, 3×3 and 5×5 in the paper) the convolution operations can look for differently sized features, making the architecture more accurate. More convolutions make the network more computationally demanding, and thus the inception module has 1×1 convolutions before the different sized kernels, to limit the input channels sizes, making the convolutions less computationally demanding [33]. A Inception module is shown in figure 15.

Finally, depthwise separable convolution performs first channel-wise (depth dimension) spatial convolution and is then followed by 1×1 convolution [6].

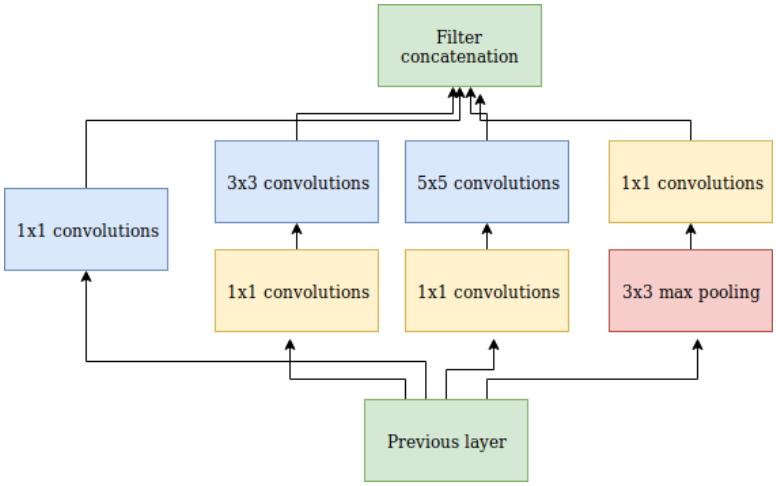

Figure 15. Inception block, figure modified from [33].

3.1.5. Hyperparameters

The layer parameters can be divided into fixed and learnable parameters based on if the parameters change when CNN is being trained. The fixed parameters are the kernel size and stride. The learnable parameters are the weights of the convolution layer's kernel. The kernel size and the weights determine what the operation returns. A convolution kernel has three parameters which are the size of the kernel, the stride and the weights. Pooling layers have only fixed parameters which are the size and the stride. The kernel size determines how many neighbouring values affect the operation, e.g. with kernel size 3×3 eight neighbouring pixels (nine in total) affect the outcome value, being it convolution in convolution layers or pooling operation in the pooling layers. Stride is used to adjust the overlap of the operations by determining how many pixels the kernel moves while sliding over the input, in cases where the kernel size is smaller than the input.

The weights are the most important part of the convolution kernel because it determines which kind of shapes are looked in the kernel sized area of the

input. As the weights are randomly initialized at first, they look for arbitrary shapes, but as the weights are adjusted in the learning part, the kernels learn to look for meaningful shapes from the input. The stride can be used on both convolution and pooling layers for downsampling the input size. Activation layers only parameter is the activation function.

3.2. Training

The objective of training the CNN is to change the convolution layer's weights and other learnable parameters so that the CNN can output the wanted results, i.e. the prediction of the correct class of the given input. The weights, which are commonly initialized randomly at first, are the parameters the network learns by optimizing the error of the network. The error of the network is the difference of the prediction (class) and the correct class that the input belongs. As such, the network needs vast amounts of correct input-output pairs, e.g. image and its correct label, for training these weights when starting from a random state.

The basic building blocks for optimizing the error of neural networks are the optimization function, its parameters, and the backpropagation method. The backpropagation is a method used to calculate all the gradients (the change of the weights) from the output of the network to back to the input, using the differential chain rule [23 p.200-201]. The most used optimization algorithms are stochastic gradient descent (SGD) and its variants like SGD with momentum, RMSprop, AdaGrad and Adam [23 p.290-305]. The SGD updates the gradient values calculated by the backpropagation, by minimizing the error of the classification by comparing the output of the network (category class) to the real class given to the network. One important parameter for the SGD is the learning rate, which is used to give boundaries for how much the optimizer changes the weights per run, which affects how fast the CNN learns. Too small learning rate and the error decreases very slowly, and too high learning rate and the error might not decrease at all.

Some of the problems which can appear in training are the overfitting and vanishing gradient problems. Overfitting means that the weights learn the training-set too accurately, and the network loses its generalisability when presented with new unseen inputs to predict. Overfitting is usually the result of too small or poorly selected training-set or an architecture that has too much capacity (learnable parameters) for the task complexity it tries to solve. Vanishing gradient is a problem which causes trainability issues for the upper layers of a deep network. The vanishing gradient occurs when deep networks use *sigmoid* or *tanh* activation functions. The *sigmoid* and *tanh* activation functions map the activation values to a range of 0 to 1. Therefore, convolution outputs are 'squeezed' to the narrow range of 0 to 1. After stacking multiple of these activation layers in the architecture, the high output values in the upper layers have a minuscule influence on the network's output. When the layers influence for the output is minuscule, so is the error caused by the layers. The error of the upper layers diminishes to zero, and the gradients can not be calculated, which means the weights of the layers are not changed. Vanishing gradients can be avoided by using activation functions

that map the values on border range than the *sigmoid* or *tanh* functions. One such activation function is the ReLu.

3.2.1. Transfer learning

Transfer learning can be seen as a technique to take existing networks with learned weights on some domain (e.g. network trained with ImageNet) and use the network to some other relatively similar task, like extracting features from different image sets. The transferability of the weights is limited by the difference of the task the weights were trained on, and the task the weights are transferred. Usually fine-tuning for the used domain is needed. Fine-tuning means training only the last layers of the network, which is usually much faster than training the whole network. It has been shown that pre-trained networks can learn general enough features to be used in transfer learning for other tasks [2, 37, 3]. The advantages of using a trained network for other tasks are the lesser amount of data and time needed to train the network. Training a network from the beginning needs a great deal of training data, which might not be available in some specialized cases, for example, medical imaging.

3.2.2. Available datasets for CNN training

There is an increasing number of publicly available labelled datasets, which can be used in image classification, object detection and recognition tasks, like Open Images [38], ImageNet [39], PASCAL VOC [40] and CIFAR [41] datasets. Probably the most used dataset is the ImageNet and its subset, which is used in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). ImageNet dataset contains 14 million images from 21 thousand classes. ILSVRC subset contains 1.2 million categorized images with 1000 different classes from the ImageNet dataset [42]. ImageNet datasets contain images of objects ranging from everyday items to different species of animals. Because of the popularity of the ImageNet dataset, there is a lot of CNN architectures published with pre-trained weights which are trained with the ImageNet dataset. As is the case with the used Xception architecture, which has readily available framework (Keras) implementation with weights trained with the ImageNet dataset. CNN architecture, which was trained with ImageNet, was selected because of the availability of these architectures. These architectures have been shown to get comparable results with the ILSVRC dataset, which is a de-facto benchmark test for new CNNs.

3.3. Xception

Xception is a CNN architecture which novelty was the adaptation of modified depthwise separable convolution layers. Xception was developed in Google [6], and it is based on the previously published Inception [33] architectures, also developed in Google. In Xception architecture, the Inception modules were changed

to the modified depthwise separable convolutions, and hence the name of 'Extreme Inception'. Xception reached 0.790 Top-1 accuracy on ImageNet in 2016. Now (as of writing in 2019) the networks with similar parameter count achieve 0.826 Top-1 accuracy [43]. Xception was chosen to be the CNN architecture of study, because of its good performance on the ImageNet dataset, and the availability of the pre-trained models with the ImageNet. Also, the readily available implementations of the architecture allowed for straightforward runs through different layers of the architecture.

Xception architecture is illustrated in figure 16, where the separable convolution layers are the modified depthwise separable convolution layers. Xception architecture also includes the residual layers and batch normalization. Xception architecture can be seen to be split to 14 different blocks which are separated by the residual layers. The layers in the implementation also follow the 14 block separation in their naming, by grouping the layers by the corresponding block they belong.

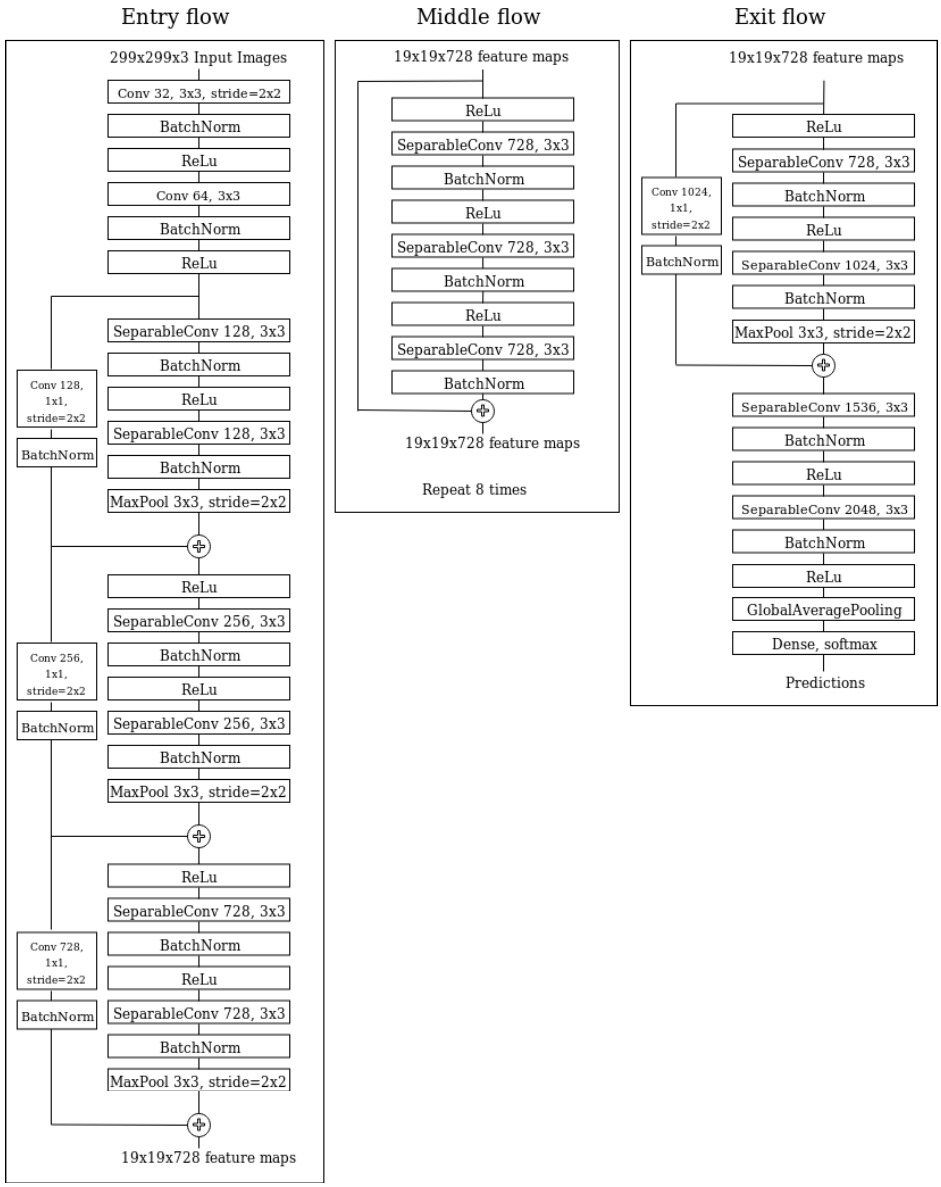

Figure 16. Xception architecture, modified from [6]. The input images are first feed through the entry flow and then through the middle flow, which is repeated eight times, and finally through the exit flow. The logistic layer softmax produces the category predictions of the input image.

4. FEATURE DIMENSION REDUCTION

High dimensional data adds complexity for processing data [9 p.237]. The data complexity by high dimensions is usually referred as 'curse of dimensionality', a term coined by Bellman [44], which describes the exponentially increasing cost of algorithms with the growth in the number of variables (dimensions) [45]. Dimension reduction (DR) is a technique used for alleviating the problems associated with high dimensions. The basic principles of dimension reduction are the concatenation of most relevant features from the high number of available and insignificant features. This can be formalized as:

$$F(x) : R^d \rightarrow R^{d'}$$

where DR is the mapping F which maps the high-dimensional feature space R^d to a lower-dimension feature space $R^{d'}$ [9 p.238].

Various clustering approaches are based on the assumption that there is a higher number of data points (N) in the dataset than feature variables (dimensions, d) in a single data point. Thus, dimension reduction can be useful in situations where the feature variables of a given data point are higher than the number of data points being clustered ($N < d$) [9 p.237]. The difference in clustering algorithms with high dimensional features can also be seen in experiments that compare the results of AC and HDBSCAN. Dimension reduction can also be used to visualize high dimensional features, by compressing the features to two or three dimensions. DR has limitations, as the process of which features to keep and which to discard is usually done via optimization of a cost function, which causes loss of information [9 p.238].

Dimension reduction algorithms can be categorized into linear and non-linear algorithms. Examples of linear algorithms are PCA [46] and ICA [47], while UMAP [48], LLE [49] and autoencoders [50] are examples of non-linear dimension reduction algorithms. The algorithms selected to be used in this thesis are the linear principal component analysis (PCA) and the non-linear uniform manifold approximation and projection for dimension reduction (UMAP). PCA is probably the most commonly used linear dimension reduction method, while UMAP is a novel non-linear method (2018). UMAP is also great at visualizing high dimensional data [50] (also, see figure 17). The algorithms are introduced in more detail in the next sections.

Dimension reduction is usually used as a preprocessing step with high-dimensional feature spaces, which the output features of different layers are. For example, some of the layers can have output features with size $147 \times 147 \times 128$, which results in features with over 2.7 million dimensions. However, as DR causes loss of information, the features in this thesis are also clustered without dimensionality reduction.

4.1. PCA

Principal Components Analysis (PCA) is a statistical method which can be used in data reduction, data compression and dimension reduction by finding a subset

of features which represent the majority of the data. PCA works by finding the single best subspace ($R^{d'}$) of the original feature space (R^d) within the criterion of least-square error [51]. This is achieved by searching orthogonal directions which maximise the variance of the dataset, assuming the feature-space is linear. Hence PCA can have problems with non-linear feature spaces. The search for orthogonal directions can be formalized as an error minimization function J which minimises the representation error E as follows (modified from [51])

$$J_{PCA} = E\left\{ \|\boldsymbol{x} - \sum_{i=1}^{d'} (\boldsymbol{w_i}, \boldsymbol{x})\, \boldsymbol{w}_i\|^2 \right\} \tag{11}$$

where orthonormal directions are vectors $\boldsymbol{w}_i$ and $\boldsymbol{x}$ is a matrix of dataset features in original d dimensions. Principal components (PCs) are derived from the orthonormal directions as $PC_{i,\cdots,d'} = ((\boldsymbol{w}_i, \boldsymbol{x}), \ldots, (\boldsymbol{w}_{d'}, \boldsymbol{x}))^t$. The PCs with maximum variances are selected to represent the data points in the target dimension d' by selecting d' number of PCs, in order that PC_1 explains most of the variance of the dataset and PC_2 the second-most, etc. [51]. If the wanted dimension parameter is not specified, the PCs are generated for all the original dimension. This can be helpful when deciding to which dimensions the dataset is reduced, by looking how much variance each PC contains. For example, if the dataset being reduced contains features with 100 dimensions and the first two PCs contain 50% of all the variance and the other 98 PCs contain evenly spread out the last 50% of the variance, it could be useful to only use the first two PCs to represent the dataset in reduced dimension. However, the optimum number of PCs is highly application dependent.

4.2. UMAP

Uniform manifold approximation and projection for dimension reduction (UMAP) is a non-linear dimension reduction method, based on manifold learning. UMAP does dimension reduction by finding a low dimensional embedding of the data that approximates an underlying manifold [48]. The inner workings of UMAP in dimension reduction are based on theoretical frameworks of Riemannian geometry and topological algebra, which can be further inspected in the original paper [48].

The simplified principle of UMAP can be described in two steps, one where k-neighbour based graph from the dataset is constructed and other where the low-dimensional layout of the graph is computed, which represents the dimension reduced version of the dataset. The strengths of UMAP for visualization come from the ability to pack similar points from the higher dimensions closer together in two dimensions, and the ability to keep enough global structure to have visualizations where there is enough separation between dissimilar points. This is illustrated in figure 17, which compares how the dimension reduction done with UMAP and PCA generates different plots. Both were used to reduce the same 2048 dimensional features to two dimensions.

UMAP implementation [52] has a multitude of hyperparameters, which can be used to fine-tune the algorithm. The parameters for the basic usage are the

Figure 17. UMAP and PCA mapping in two dimensions. 1000 features with a dimension of 2048 were reduced to two dimensions with UMAP and PCA, then clustered with AC. The colours represent the labels of for each cluster.

number of neighbours (k) used in the k-nearest neighbour search, the target dimensionality (d') of the dataset, the minimum distance which controls how tightly the points are packed, and the metric to be used in the calculation of the pairwise distance matrix. Other parameters were left as default values, unless mentioned specifically.

One notion of UMAP regarding this thesis use case is the optimization of the low-dimensionality layout, which is based on the SGD optimizer. SGD uses sub-sampling for the optimization. This effectively means that the order of features feed to UMAP affects how the lower-dimensionality is constructed. As stated in the original paper *'Since UMAP makes use of both stochastic approximate nearest neighbour search, and stochastic gradient descent with negative sampling for optimization, the resulting embedding is necessarily different from run to run, and under sub-sampling of the data.'* [48]. To get reproducible results when using UMAP as DR step before clustering, the features need to be feed in the same order for each run, and the sampling needs to have the same random seed number.

5. DATASETS

The clustering results and features extracted by the CNN highly depend on the images in the datasets. Eight image sets were selected to be used as evaluation datasets. The evaluation datasets are selected within the assumption that the datasets contain images with enough differences that the extracted features, by some of the layers, results in clusters forming. The requirements for the datasets depend on the selected validation metric. When external clustering validation is used, the dataset being clustered needs to have labels on the data. The dataset labels are compared to the labels given by the clustering algorithm. In this thesis, the selected clustering validation metrics were the external NMI and ARI scores, and thus the dataset being clustered needs to have labels.

The datasets were selected for testing the layers feature extraction capabilities based on different clustering tasks. The selected datasets and their clustering tasks can be seen in Table 1. Next subsections introduce the datasets more thoroughly.

Table 1. Properties of the evaluation datasets

Dataset Name	**Clustering Task**	**Img. Size**	**Classes**	**Images**
ImageNet10	Object detection	Varying	10	500
flower&qr	Object detection	256×256	2	500
Tools	Object detection	640×480	7	560
overcast&lowlight	Scene detection	256×256	2	500
cab&chicken, sigma2	Object / blur detection	Varying	2	200
cab&chicken, sigma3	Object / blur detection	Varying	2	200
cab&chicken, sigma5	Object / blur detection	Varying	2	200
cab&chicken, sigma9	Object / blur detection	Varying	2	200

5.1. *ImageNet10*

ImageNet10 is a dataset with images from 10 classes from the ILSVRC 2012 validation set, each class containing 50 images. As the used CNN was trained with ImageNet dataset, these classes selected from the validation image set represent the baseline dataset for extracting features which should generate good clustering results in the deeper layers. The selected categories from the ILSVRC 2012 validation set are *jigsaw puzzle, dial telephone, ground beetle, analog clock, cup, hay, French bulldog, great grey owl, bearskin* and *steel drum.*

5.2. *flower&qr*

The second dataset is the *flower&qr* dataset, that is constructed from in house datasets used for other projects. The *flower&qr* dataset was created by selecting images from two datasets, one category having images similar as in the CNN's

training dataset, while the other category has images which are not explicitly used in the CNNs training set. The flower images should generate distinguishable features with the CNN because the used training dataset (ImageNet) contains images of flowers. The second category in this dataset was selected to be compromised of images augmented with a QR-code image on top of the image because QR-code images are not explicitly trained with the CNN. As to test if the extracted QR-code image features can be separated from the flower images. The *flower&qr* dataset has two categories, each one having 250 images.

5.3. *Tools*

The dataset *Tools* was published by the authors of [3], which they used in their Xception feature extraction and clustering pipeline. The CNN architecture is the same as used in this thesis, and thus their dataset can be used as a validation method for the pipeline constructed in this thesis. The Tools dataset contains seven object categories with images of tools like allen key, clamp, driver, flat and images from pens, screws and USB sticks. The category images are constructed with five lighting and background conditions. All of the conditions and the objects are used as one mixed image set, having seven classes based on the object category, containing 560 images.

5.4. *overcast&lowlight*

The *overcast&lowlight* dataset contains images from two scene categories, a low light and overcast categories. The first category is constructed from low light images, e.g night time images where the major part of the image is dark while having some light sources in the image, of different things like objects and scenes, while the second category contains overcast (cloudy skies) images which are taken from objects and scenes in outdoor environments. The low light and overcast images can both have objects similar to the CNN's training dataset, and thus the extracted features can be similar to images from both of the categories, as the dataset is labelled based on the scene of the images and not by the objects in the images. This dataset tests if the CNN's features from some of its layers can be used to separate scene images. The *overcast&lowlight* dataset has 500 images, 250 on both categories.

5.5. *cab&chicken*

The *cab&chicken* datasets are constructed from images in the cab and prairie chicken categories from the ILSVRC 2012 validation set. The datasets also contain blurred versions of the cab and prairie chicken images. The images were blurred with Gaussian blur, with a kernel size of 31×31 and with sigma values 2, 3, 5 and 9. The *cab&chicken* datasets are separated and named based on the value of the sigma used in the Gaussian kernel in the dataset. The *cab&chicken*

datasets have two clustering targets, cluster images to category groups or blurred and non-blurred image groups. These datasets are used to test if some of the features from the layers can be used to cluster images to groups based on blurriness. Additionally, these datasets are used to test how different amounts of blur in the images affect the clustering to the image categories. The datasets have 200 images, 50 for each group and 100 per class.

6. IMPLEMENTATION AND RESULTS

6.1. Experimental Setup

Within the increased usage of CNN based systems, there has been growth in readily available software packages and frameworks for developing and training CNNs. Few of the most used frameworks are TensorFlow, Torch, Keras, Theano and Caffe, which TensorFlow is probably the most popular. All of the mentioned frameworks have application programming interfaces (APIs) or bindings to Python, which made the selection of the implementation programming language obvious. From the frameworks, TensorFlow was selected to be the baseline framework with supplementary high-level APIs from Keras. Additional libraries used in the implementation are machine learning library scikit-learn and the Python implementations of the UMAP and HDBSCAN algorithms.

TensorFlow is a mathematics framework with a focus on machine learning. TensorFlow was developed in Google, and made available for the public in 2015 [53]. Keras is a neural networks library which is written in Python, allows for fast prototyping, and can be used on top of TensorFlow. Keras library also has an implementation of Xception model with pre-trained weights trained on the ImageNet dataset [54]. Scikit-learn is a machine learning library for Python [22] which has the implementations for the AC clustering, PCA dimension reduction and the NMI and ARI validation score implementations, while the UMAP and HDBSCAN have implementations provided from [52] and [15]. The used hardware for the implementation was desktop PC with Ryzen 7 2700X 8-core CPU with 32 GiB of onboard RAM and 1050ti GTX GPU.

6.1.1. Pipeline

The implementation pipeline is shown in figure 18. The figure shows an example of a two-category clustering workflow. The images are first loaded and preprocessed by Keras preprocessing implementation and then run through Xception's layers to the selected layer. The naming for the layers follows the naming schema of Xception's Keras implementation. The output from the selected layer are the features for the corresponding image. The features are either clustered straight on or run through dimension reduction. When the features are clustered, the corresponding image which the features were extracted is given the same label as the clustering gave to the features. Validation metrics are used to measure the correctness of the clustering. When a layer is changed, clustering and dimension reduction parameters are kept the same on the runs with each layer. If the clustering, dimension reduction, or other parameters are changed, all the layers are run with the changed parameters.

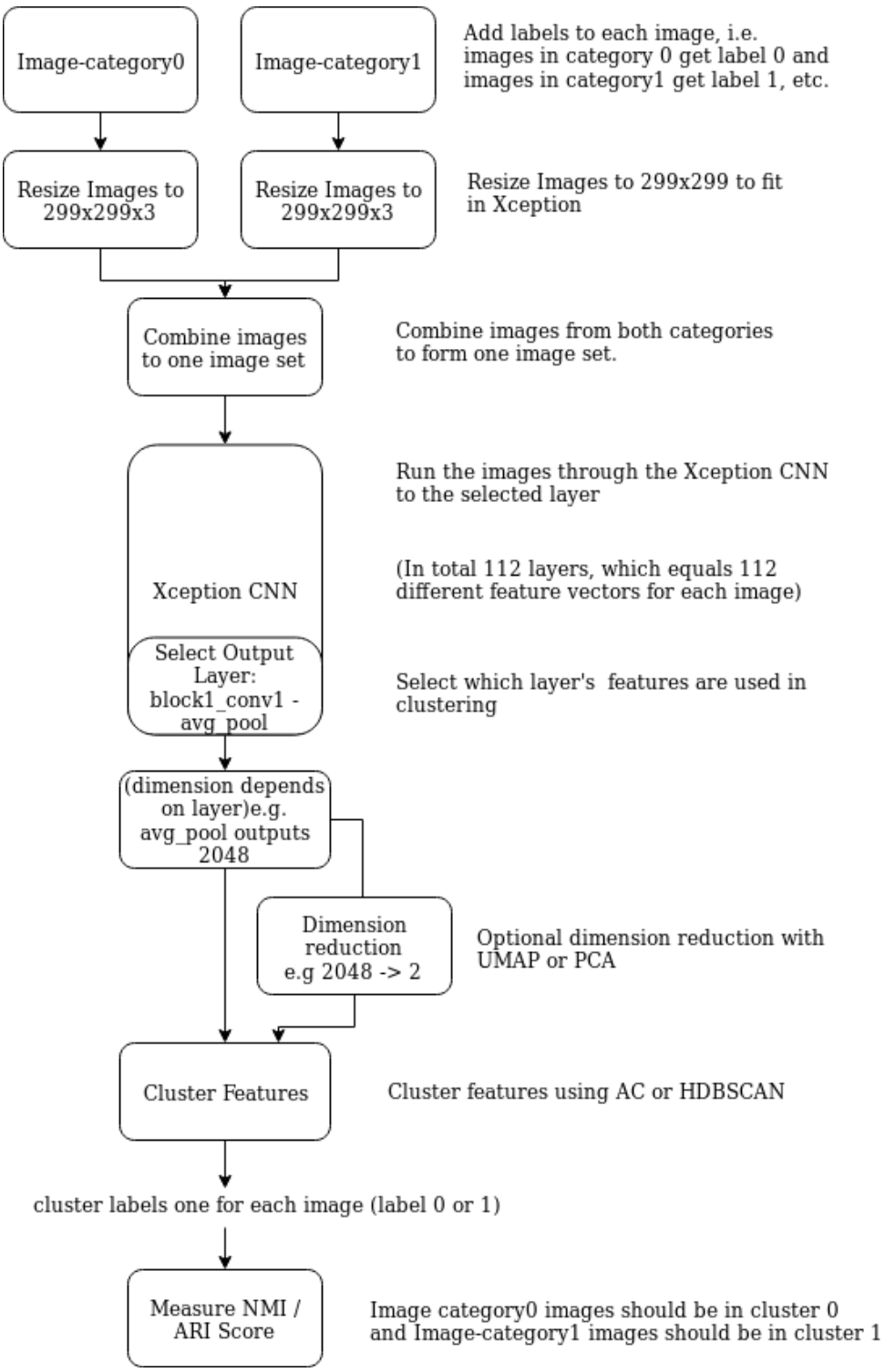

Figure 18. The implementation pipeline. This example shows the implementation workflow when clustering dataset with two categories. Images in the categories are first labelled. The images are preprocessed and combined for one image set that is sequentially run through the CNN to extract the image features with the selected layer. The image features are clustered, and the cluster labels are compared to the label given to the image.

6.1.2. Hardware limitations

The features extracted with Xception's second block of layers have large dimensions. Xception architecture's block 2 contains convolution kernels with 128 dimensions with the input height and width being 147, which means that the total memory consumption per image's features can be calculated as follows:

$$MemConsPerImg = float32bit * width * height * kernelSize$$

this results, assuming 32 bit floating points, to 88.51 Mb or 11.06 MB per image. The memory consumption in the actual implementation doubles because the features are converted to 64 bit floats in the clustering algorithms, resulting to 22.12 MB memory usage per image.

6.2. Quantitative results

First, the evaluation datasets were run without dimension reduction and with agglomerative clustering, using the number of clusters parameter according to the used dataset, leaving the rest as default values. The NMI scores with feature dimension reduction and HDBSCAN clustering are shown in the appendices. Table 2 shows the NMI median values and standard deviation (σ) per layer block (layer blocks, see section 3.3) for the first four datasets. The NMI median values for each block in the *cab&chicken* datasets are shown in Table 4.

Table 2. NMI value medians and standard deviations in layer groups (blocks) for datasets

	ImageNet10		*flower&qr*		*Tools*		*overcast&lowlight*	
	Median	σ	Median	σ	Median	σ	Median	σ
Block1	0.12	0.01	0.28	0.19	0.10	0.00	0.55	0.06
Block2	0.12	0.01	0.68	0.15	0.12	0.01	0.47	0.10
Block3	0.20	0.01	0.82	0.12	0.12	0.01	0.15	0.08
Block4	0.21	0.04	0.80	0.18	0.08	0.03	0.20	0.15
Block5	0.26	0.07	0.83	0.39	0.08	0.01	0.05	0.20
Block6	0.31	0.09	0.07	0.42	0.07	0.00	0.03	0.01
Block7	0.28	0.09	0.81	0.41	0.04	0.03	0.04	0.01
Block8	0.27	0.03	0.05	0.01	0.07	0.03	0.04	0.02
Block9	0.44	0.09	0.96	0.49	0.02	0.02	0.07	0.13
Block10	0.42	0.14	0.81	0.44	0.04	0.02	0.03	0.07
Block11	0.53	0.09	0.95	0.48	0.03	0.02	0.17	0.08
Block12	0.57	0.09	0.94	0.30	0.03	0.01	0.24	0.07
Block13	0.42	0.12	0.06	0.42	0.02	0.02	0.04	0.03
Block14	0.87	0.12	0.93	0.46	0.25	0.17	0.15	0.16

As seen in the Table 2 the best NMI scores usually came from layers in the last block, block 14. The exceptions were the *flower&qr* and the *overcast&lowlight*

datasets. *flower&qr* dataset had high median values from multiple blocks, and the best value came from the block 9 at 0.96, while the last block had a value of 0.93. *overcast&lowlight* dataset had significantly better scores from the first two blocks than from the last block. The layers which features were clustered with the highest NMI scores for each dataset can be seen in Table 3. The *overcast&lowlight* and the *cab&chicken* (blur) datasets had highest scores from layers quite early in the architecture, when compared to the other datasets with category clustering (exception being *flower&qr* dataset). The next subsections show the layer NMI scores more in depth, on a dataset basis.

Table 3. Highest NMI scores of each dataset

Dataset	**NMI score**	**Layer**	**DR**	**Clustering**
ImageNet10	0.95	avg_pool	UMAP	AC
flower&qr	1.0	block5_sepconv1[1]	None	AC
Tools	0.67(0.03)[2]	avg_pool	UMAP	HDBSCAN
overcast&lowlight	0.70	block5_sepconv1_act	PCA	AC
sigma2(Category)	1.0(0.0)[2]	block10_sepconv2_act[1]	UMAP	HDBSCAN
sigma3(Category)	1.0	block12_sepconv1[1]	None	AC
sigma5(Category)	1.0	block12_sepconv1_bn[1]	None	AC
sigma9(Category)	1.0	block14_sepconv1_act[1]	None	AC
sigma2(Blur)	0.80	block3_sepconv1_bn	PCA	AC
sigma3(Blur)	0.90	block3_sepconv2	PCA	AC
sigma5(Blur)	1.0	block5_sepconv1	PCA	AC
sigma9(Blur)	1.0	block3_pool[1]	PCA	AC

[1)]The first layer in the architecture with the maximum score is shown if there is multiple layers with score of 1.0.
[2)]The noise group percentage is shown in brackets with HDBSCAN clustering.s

ImageNet10

The *ImageNet10* dataset had a trend on increasing the maximum NMI values when deeper layers were selected. However, on some of the layers, the NMI value declined (see figure 19). Majority of the NMI value drops came from the third separable convolution layers (sepconv3), either on the batch normalisation, convolution or activation parts of these layers. The best scores came from the last five layers, having NMI values over 0.85. The network was trained to separate these image categories from each other, and thus, the increased separation of these image features increased as deeper layers were selected.

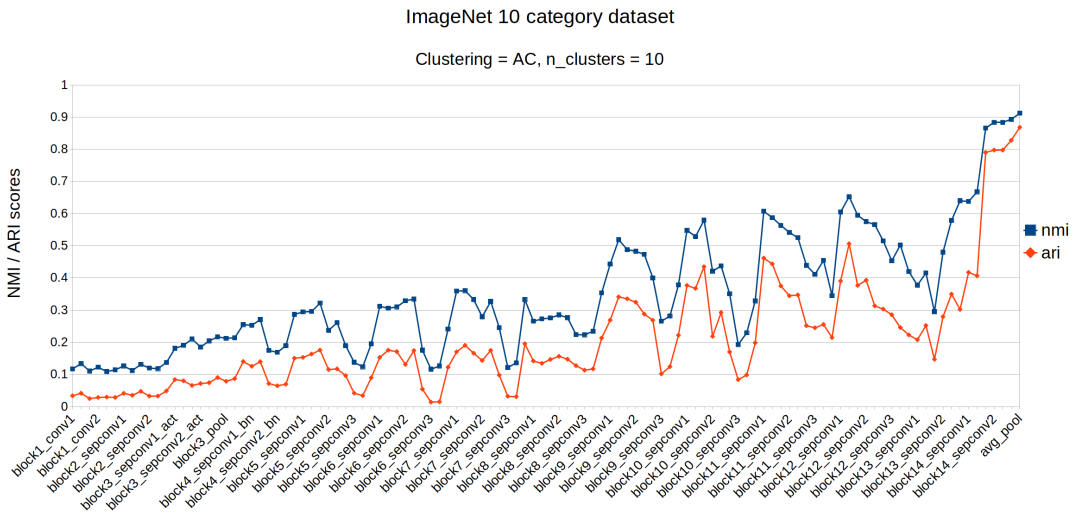

Figure 19. *ImageNet10* dataset clustered with AC, the number of clusters parameter set to 10, and other parameters are left for default. The majority of the drops in NMI values came from the sepconv3 layers. The features from the average pooling layer gave the best NMI score.

flower&qr

The *flower&qr* dataset results show that features extracted with the seventh layer already gave distinguishable features for clustering this dataset (see figure 20), as the resulting NMI value for the layer was 0.91. The *flower&qr* dataset is probably easier to cluster with its two categories than the 10 category *ImageNet10* dataset, and thus high NMI values can be achieved with the features form layers quite early in the architecture. The NMI values drop on the sepconv3 layers, similar as in the *ImageNet10* dataset, but more drastically.

Tools

In the *Tools* dataset, the NMI value stays below 0.1 on the majority of the layers, as 90 layers out of 112. On the entry flow layers (layers in blocks 1 to 4), the value stays below 0.15 and only on the last four layers does the NMI value increase being 0.54 on the last layer (same value as in the article [3]) and around 0.39 on the three previous layers (see figure 21). The scores seem to indicate that layers other than the last four could not extract features which could be clustered to corresponding groups. The specific features from the deeper layers were more prominent in separating the categories from each other than the general features extracted from the upper layers.

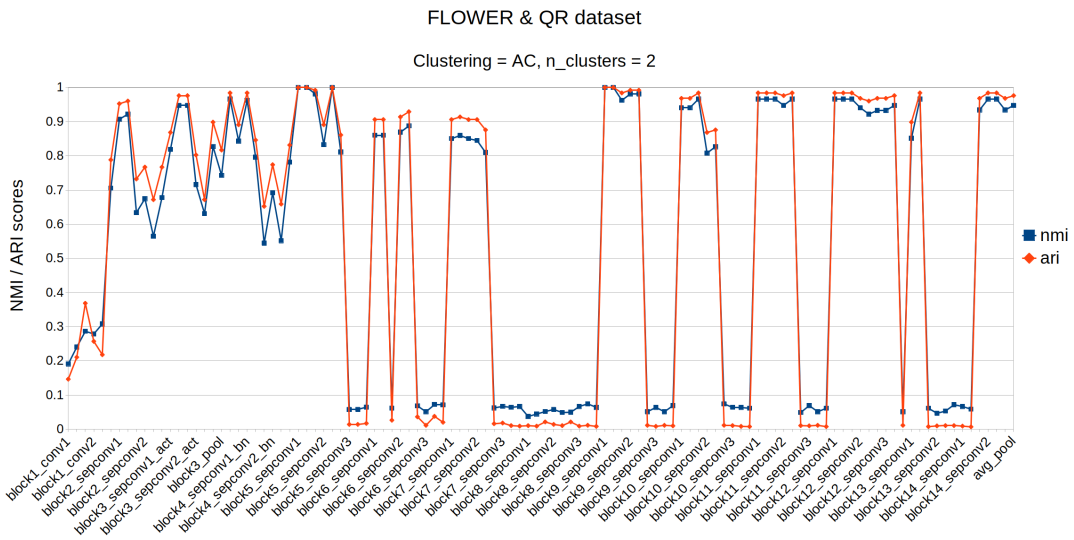

Figure 20. *flower&qr* dataset clustered with AC, the number of clusters parameter set to 2, and other parameters are left for default. The low NMI values come from the sepconv3 layers, except the last sepconv3 layer in block13. Features from five layers are clustered with perfect NMI score (1.0).

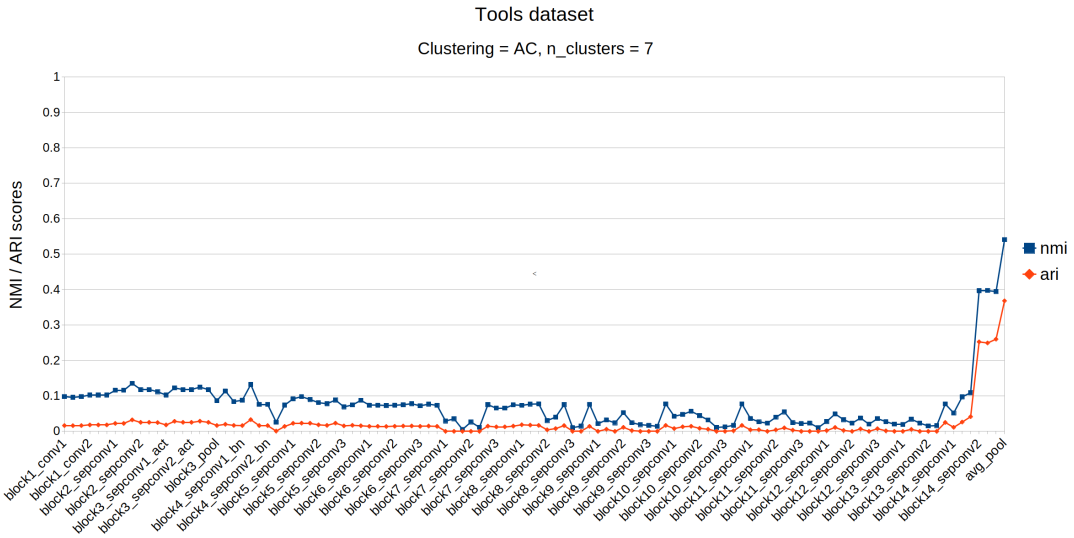

Figure 21. *Tools* dataset clustered with AC, the number of clusters parameter set to 7, and other parameters are left for default. Only the features from the last four layers gave significant clustering scores.

overcast&lowlight

The *overcast&lowlight* dataset NMI values for the first three layers are above 0.55, while the majority of the layers have NMI values below 0.50. The highest NMI value comes with features extracted on the layer block5_sepconv1_act1, having

NMI value of 0.60. The results indicate that the features extracted with the upper layers from this dataset can be used to cluster the images more accurately than with the features extracted with the deeper layers. The general features were more prominent in separating the categories in this dataset, as the deeper layers extract more specific features which could not be used to separate these two categories. The *overcast&lowlight* dataset NMI values can be seen in figure 22.

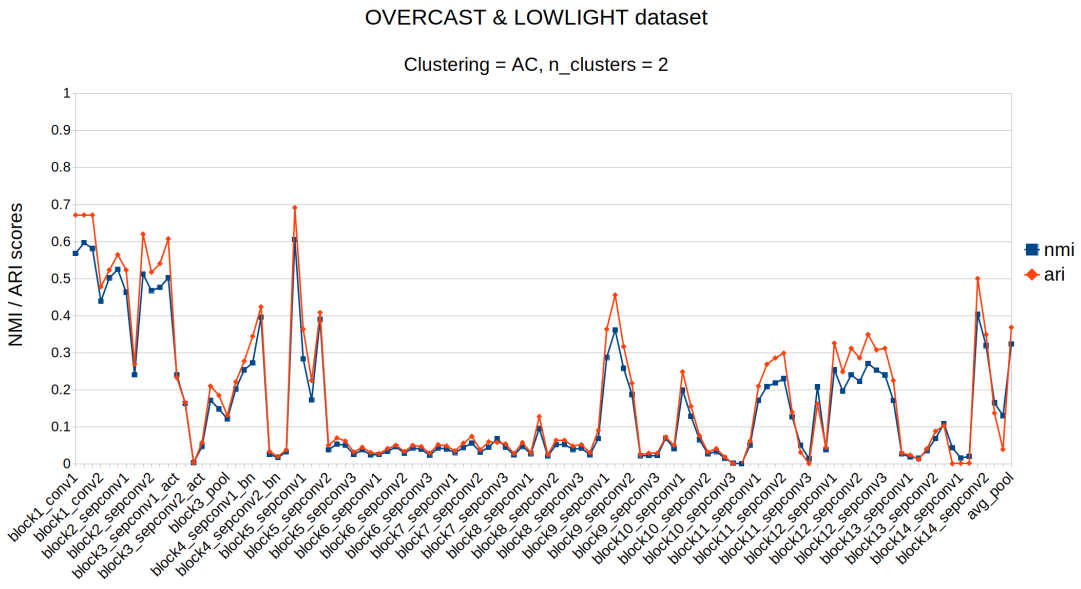

Figure 22. *overcast&lowlight* dataset clustered with AC, the number of clusters parameter set to 2, and other parameters are left for default. The general features from the first few layers give significantly higher cluster NMI values for the clustering than the features extracted with the last average pool layer (**0.57** in the first layer and **0.32** in average pool layer) with these scene images in the *overcast&lowlight* dataset.

cab&chicken

The *cab&chicken* datasets show how the layer NMI scores differ when the clustering task changes from image category clustering to blurred and non-blurred image clustering. The dataset with the least amount of blur (sigma 2) in the images did not cluster to blurred and non-blurred images, as the maximum NMI score from the whole architecture was 0.23. The image category clustering task had similar NMI score behaviour as with the *flower&qr* dataset, as seen in figure 23. The NMI score increased to 0.80 in the first 15 layers, and on the first four of the sepconv3 layers, the NMI scores dropped significantly.

The second *cab&chicken* dataset with sigma value 3 could be clustered to blurred and non-blurred image groups with some of the layers. Over 0.65 NMI scores came from the 8th, 9th and 10th layer in the architecture. Other peaks in

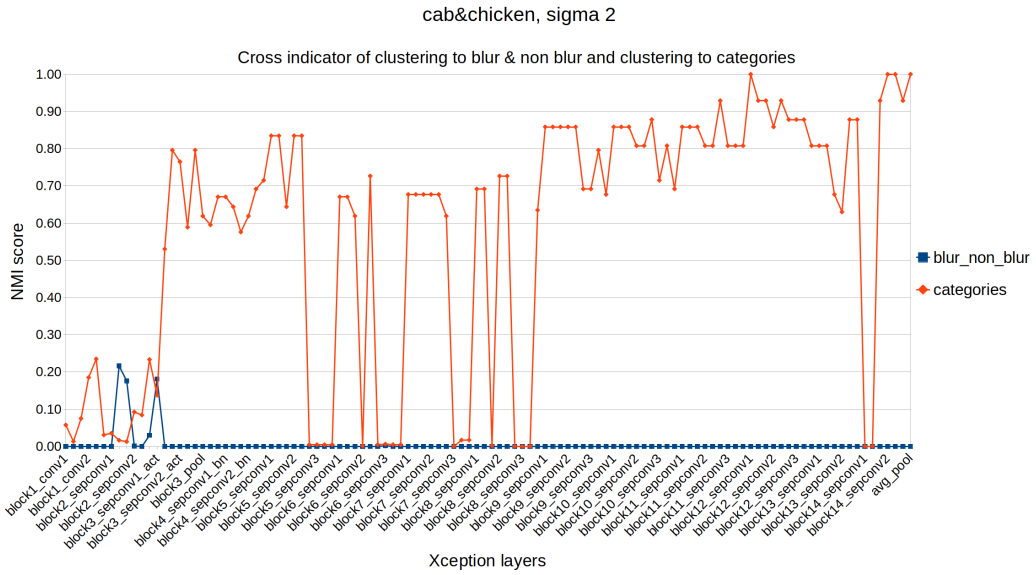

Figure 23. *cab&chicken, sigma 2* dataset clustered with AC, the number of clusters parameters set to 2, and other parameters are left for default. NMI scores when clustering to blurred and non-blurred groups (blue) and image categories (red). Images with the least blur did not group based on blurriness. The category clustering had similar NMI score profile as in the *flower&qr* dataset, with four first blocks with sepconv3 layers had significantly lower NMI scores than majority of the layers.

the NMI scores are from the pooling layers of block 2, block 3 and block 4, each having NMI scores over 0.40. Two of the layers between the pooling layers got NMI scores over 0.70 in the category clustering, which indicates that the layers switch on detecting blurriness, then image category, and then a subsequent layer can recover the blurriness again. The NMI scores for the sigma 3 dataset can be seen in figure 24. The image category clustering for the dataset with sigma value 3 had similar NMI scores as in the dataset with sigma value 2. Some differences to the NMI scores came from the layers which could be used to detect the blurriness of the images. This phenomenon can also be seen in Table 4, which lists the median NMI values of the *cab&chicken* datasets for both clustering tasks over different layer groups. The same layers do not achieve good clustering NMI scores with both of the clustering tasks.

For the sigma 5 dataset, the same 8th, 9th and 10th layers had similar NMI scores for clustering the image set based on blurriness as the dataset with less blurry images. The differences to the NMI scores came from the layers in block 3 and in block 4, which had significantly increased NMI scores from the less blurred dataset on the blurriness clustering task, shown in Table 4 when comparing the medians of block 3 and 4 between the datasets. The NMI scores for the layers with sigma 5 dataset can be seen in figure 25. From figure 25, we can see that the NMI scores for the category clustering task were lower on layers from block 1 to block 9 than in the sigma 3 dataset.

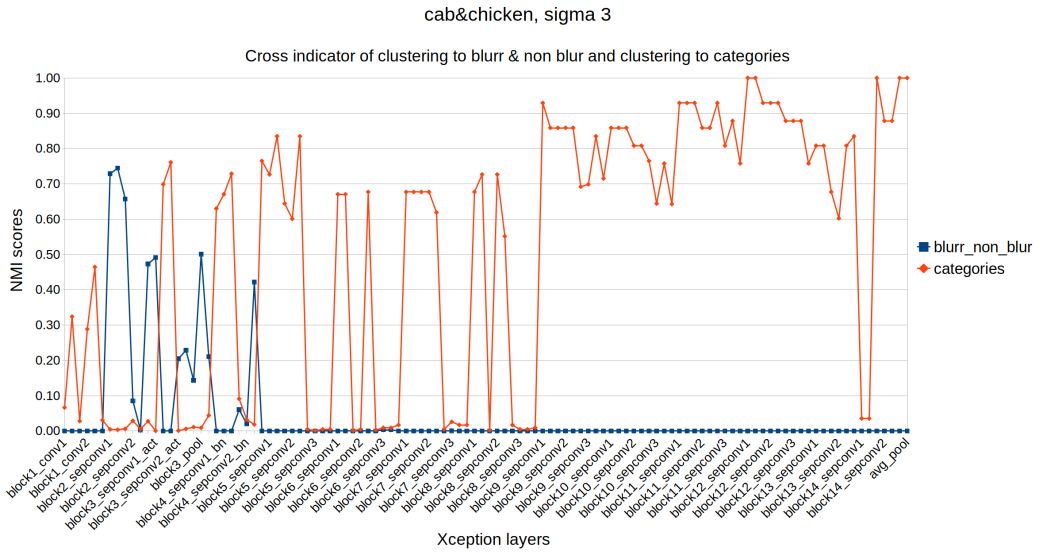

Figure 24. *cab&chicken, sigma 3* dataset clustered with AC, number of clusters parameters set to 2, and other parameters are left for default. NMI scores when clustering to blurred and non-blurred groups (blue) and image categories (red). When the image blurriness is increased to sigma 3, the images can be clustered based on the blurriness. The features from the first sepconv1 layer in block 1 are clustered by blurriness with a NMI value of 0.73.

Table 4. *cab&chicken* datasets median NMI values on both clustering tasks over layer groups in each dataset

Dataset	**Sigma2**		**Sigma3**		**Sigma5**		**Sigma9**	
Task	Category	Blur	Category	Blur	Category	Blur	Category	Blur
Block1	0.07	0.00	0.18	0.00	0.21	0.00	0.36	0.00
Block2	0.06	0.02	0.01	0.57	0.01	0.67	0.00	0.74
Block3	0.62	0.00	0.01	0.20	0.00	0.86	0.00	0.96
Block4	0.64	0.00	0.09	0.02	0.00	0.86	0.00	1.00
Block5	0.71	0.00	0.64	0.00	0.00	0.01	0.00	0.86
Block6	0.01	0.00	0.01	0.00	0.02	0.00	0.01	0.02
Block7	0.68	0.00	0.62	0.00	0.01	0.00	0.00	0.66
Block8	0.02	0.00	0.02	0.00	0.02	0.00	0.01	0.00
Block9	0.86	0.00	0.86	0.00	0.76	0.00	0.79	0.00
Block10	0.81	0.00	0.81	0.00	0.83	0.00	0.78	0.00
Block11	0.81	0.00	0.88	0.00	0.86	0.00	0.79	0.00
Block12	0.88	0.00	0.93	0.00	0.83	0.00	0.86	0.00
Block14	0.93	0.00	0.88	0.00	0.94	0.00	1.00	0.00

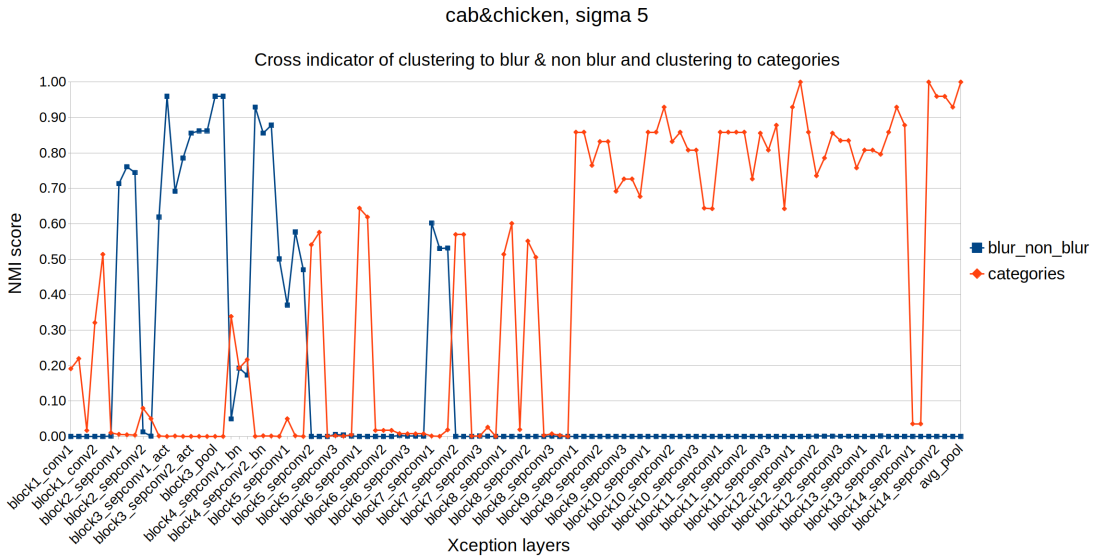

Figure 25. *cab&chicken, sigma 5* dataset clustered with AC, the number of clusters parameters set to 2, and other parameters are left for default. NMI scores when clustering to blurred and non-blurred groups (blue) and image categories (red).

The dataset with the most blurred images, had best NMI scores from layers in the block 3, 4 and 5 for the blurriness clustering task, while these same layers failed in image category clustering. However, the best NMI scores for the image category clustering task came from the layers in blocks 9 through 11, and the last five layers in block 14. Interestingly, the last five layers in the sigma 9 dataset had perfect scores on the category clustering while on the datasets with less blur the last five layers only had perfect scores on two or three of the layers. The NMI scores are shown in figure 26. The *cab&chicken* datasets (sigma 9 particularly) shows how the increased blur in the images lead to totally different clusters depending on the layer.

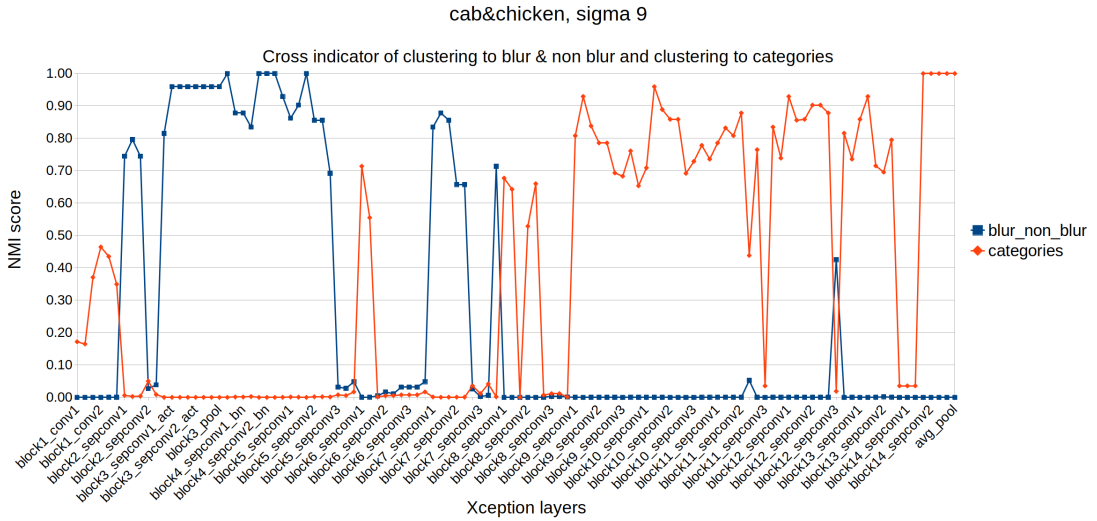

Figure 26. *cab&chicken, sigma 9* dataset clustered with AC, the number of clusters parameters set to 2, and other parameters are left for default. NMI scores when clustering to blurred and non-blurred groups (blue) and image categories (red).

6.3. Qualitative results

The qualitative results were obtained by visually examining the clusters. The layers with the maximum NMI difference between adjacent layers were selected to be inspected. The plots are derived by reducing the feature vectors to two dimensions with UMAP and plotted to a two-dimensional plane. The individual colours represent the clusters the clustering algorithm set the corresponding image. The clustering is performed with the original feature dimensions, and the features are reduced only for the plotting. The plot is a projection of the high dimensional features, so comparing the positions of the points in the two-dimensional space does not represent the real positions of the features on their native high dimension space. However, the plots can still give us some insights about the clustering if we assume that the UMAP can keep the local and global structures reasonably close to the original dimensions.

ImageNet10

In the *ImageNet10* dataset, the layers with the maximum change between them were layers *block11_sepconv1_act* and *block11_sepconv1,* where the NMI score changed from 0.33 to 0.61. Figure 27 shows the clustering of the features extracted with layer *block11_sepconv1_act* and figure 28 shows the clustering with *block11_sepconv1* features. The activation layer's features appeared to be more densely together than the features from the separable convolution layer. This could be the reason the clustering accuracy increased with the later layer's features.

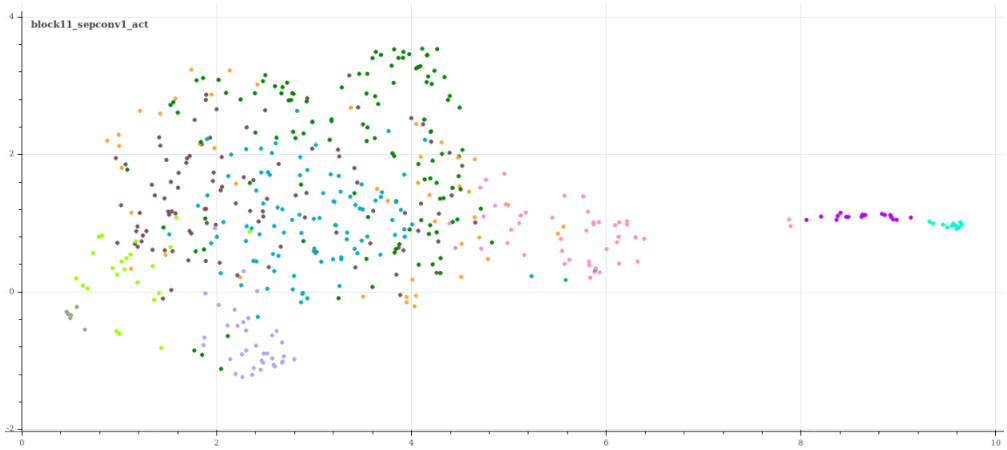

Figure 27. *ImageNet10* dataset clustered with the features extracted with *block11_sepconv1_act* layer. AC algorithm with the number of clusters set to 10. NMI score 0.33.

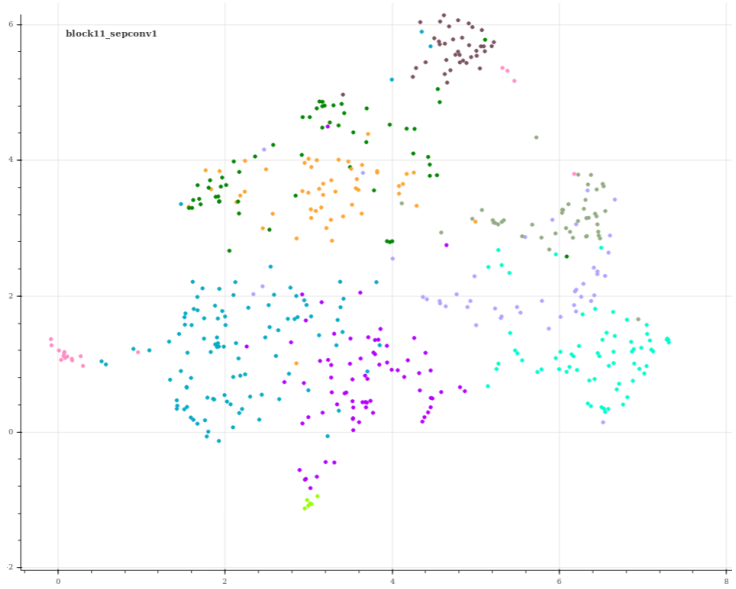

Figure 28. *ImageNet10* dataset clustered with the features extracted with *block11_sepconv1* layer. AC algorithm with the number of clusters set to 10. NMI score at 0.61.

flower&qr

The *flower&qr* dataset had many adjacent layers with high differences on the NMI values. The adjacent layers with the maximum NMI score difference were the layers *block9_sepconv1_act* and *block9_sepconv1* with NMI scores of 0.06 and 1.0, respectively. Figure 29 shows the clustering of the features from layer *block9_sepconv1_act* and figure 30 shows the clustering of the features from the layer *block9_sepconv1*. In figure 29, there can be seen two separate groups forming to the left and right edge of the plot, even though they both are coloured green as

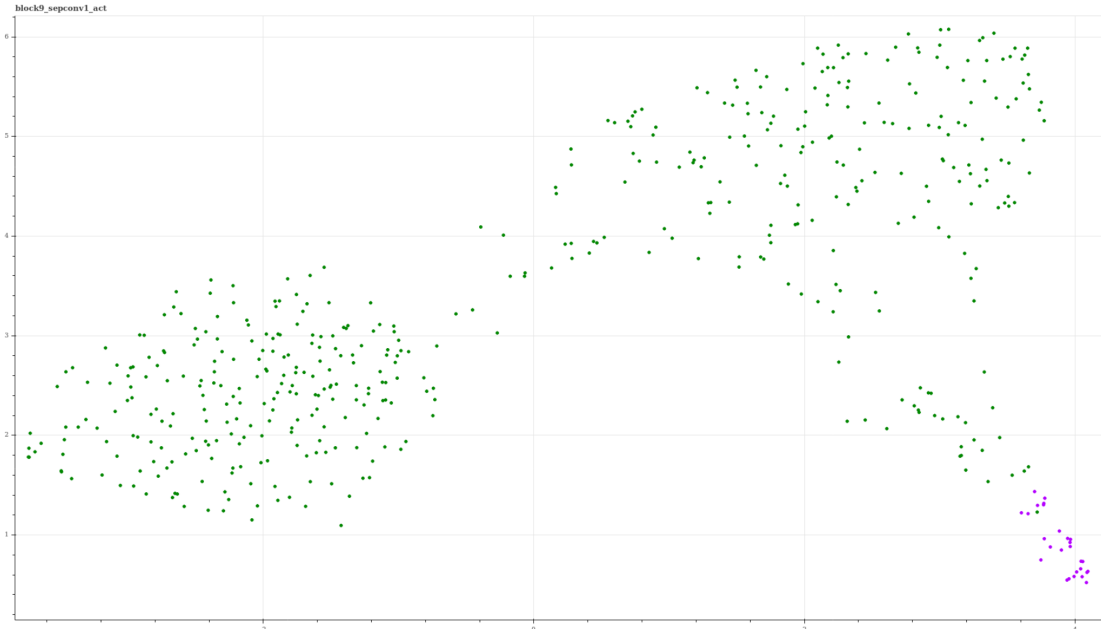

Figure 29. *flower&qr* dataset clustered with the features extracted with *block9_sepconv1_act* layer. AC algorithm with the number of clusters set to 2. NMI score at 0.06.

marked to being on the same cluster. After inspecting the images on the groups, the left side seemed to contain mostly flower images as the right side group contained mostly QR images. The reason why the majority of both categories were clustered to the same cluster could be explained by the AC algorithms way of merging single points to groups based on their pairwise distance. On the right bottom side of the plot, there is the clustering algorithm's other cluster marked with violet colour, which contains images from both categories. As explained in section 2.2, the AC merges points closest to each other to the same cluster, and so, the features in the images in the violet group have higher pairwise distances than the features in the major groups of flowers and QR images. This results for the flower and QR images being in the same cluster and low NMI value.

In figure 30, the features are clustered perfectly to their corresponding groups and the features are separated clearly in the plot. As to answer why the features changed so radically between these two adjacent layers that the NMI values increased from near zero to perfect, could be explained by the plot in figure 29. As explained above, figure 29 did contain separate groups for the flower and QR code images, but the clustering algorithm did not cluster them correctly. This could be speculated to be due to a few of the images having features which were highly different from the rest of the features, e.g. they were outliers and thus the AC algorithm broke down. However, testing this is left for future work.

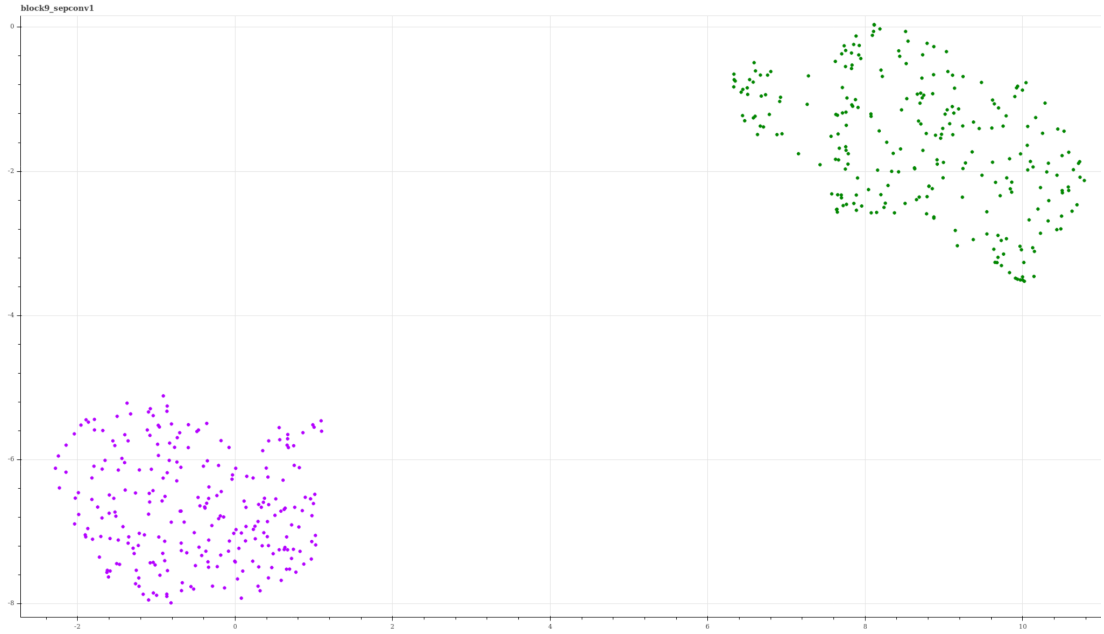

Figure 30. *flower&qr* dataset clustered with the features extracted with *block9_sepconv1* layer. AC algorithm with the number of clusters set to 2. NMI score at 1.0.

Tools

In the *Tools* dataset, the maximum NMI score difference is between layers *block14_sepconv1_act* and *block14_sepconv2*, with NMI scores of 0.11 and 0.40. The corresponding clusterings are shown in figure 31 and 32. The activation layer features formed some clusters where the dominant features were the image background, rather than the image object, and the images were grouped based on the background. In figure 31, there is an example of these object images where the clusters were formed based on the background, and not based on the image object. Both of the USB images in figure 31 are in different groups because their backgrounds are vastly different. In figure 32, with clustering on the layer *block14_sepconv2* features, the clusters are more separated from each other than in figure 31 and contain more objects from the same categories. This is probably the cause for the increased NMI score. The leftmost group in figure 32 still contains only images with the same metallic background, rather than being a group of images with the same objects.

overcast&lowlight

The *overcast&lowlight* dataset had maximum NMI change between the layers *block4_pool* and *block5_sepconv1_act*, with NMI values of 0.03 and 0.61. Figures 33 and 34 show the clusterings for layer *block4_pool* and *block5_sepconv1_act*.

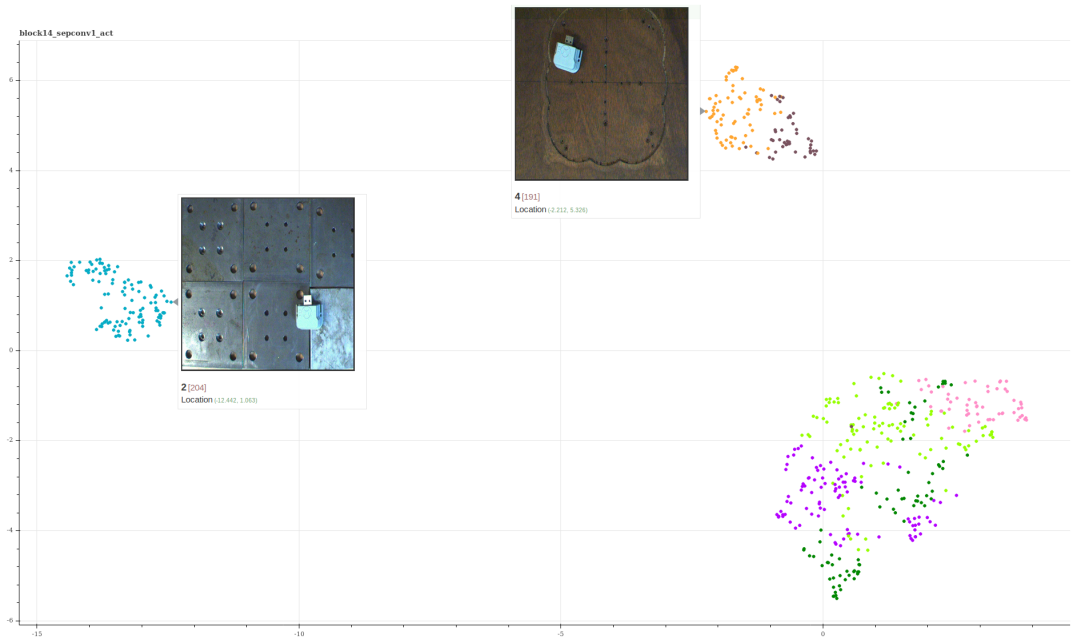

Figure 31. *Tools* dataset clustered with the features extracted with *block14_sepconv1_act* layer. AC algorithm with the number of clusters set to 7. NMI score at 0.11. Same objects (USB) with different backgrounds are clustered to different groups. Images in the figure are from the *Tools* dataset and are taken by the authors of the dataset [3].

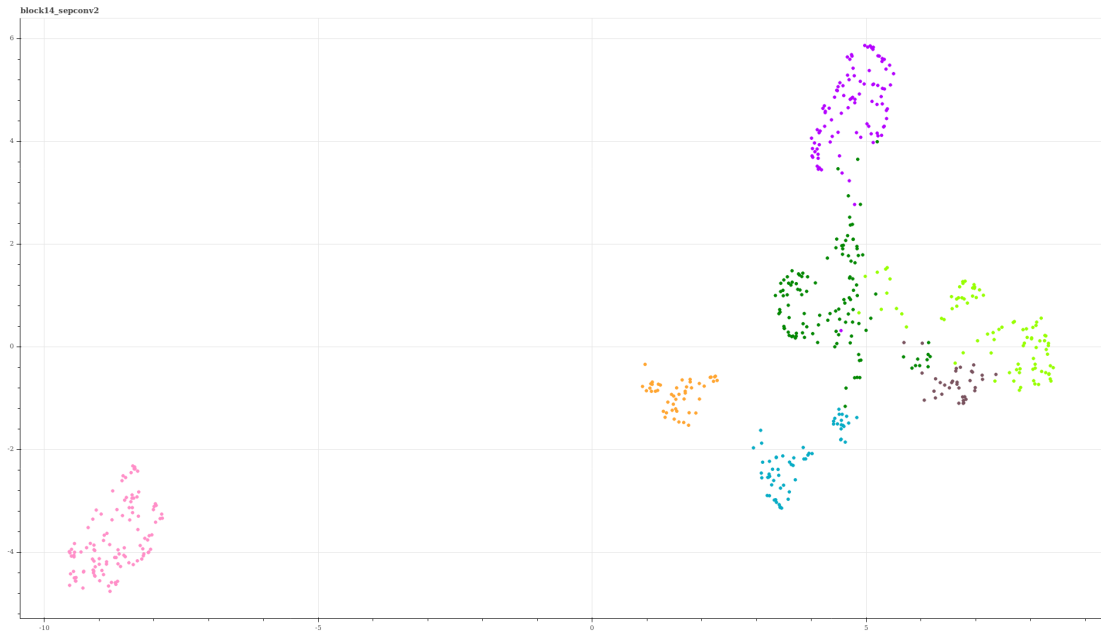

Figure 32. *Tools* dataset clustered with the features extracted with *block14_sepconv2* layer. AC algorithm with the number of clusters set to 7. NMI score at 0.4.

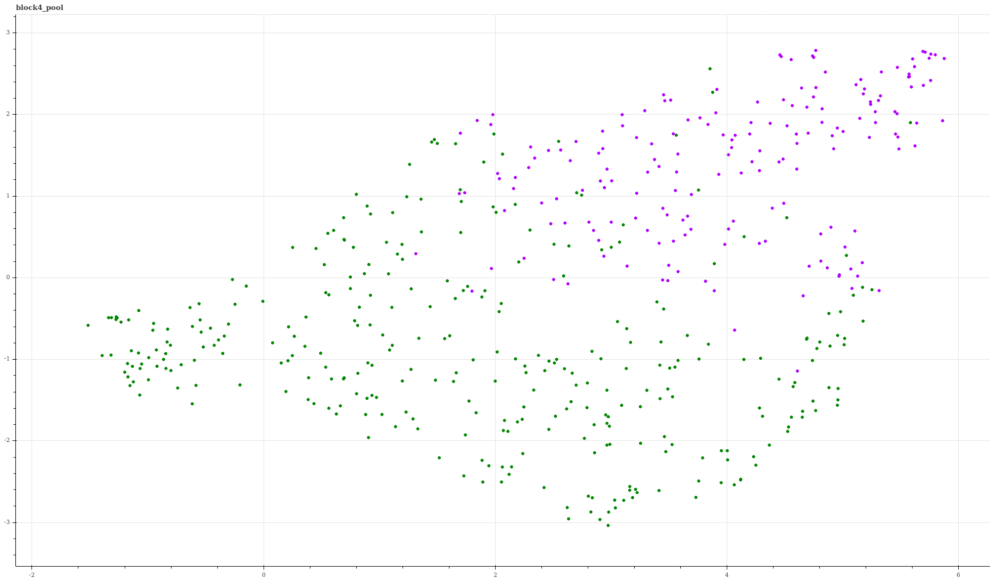

Figure 33. *overcast&lowlight* dataset clustered with the features extracted with *block4_pool* layer. AC algorithm with the number of clusters set to 2. NMI score at 0.03.

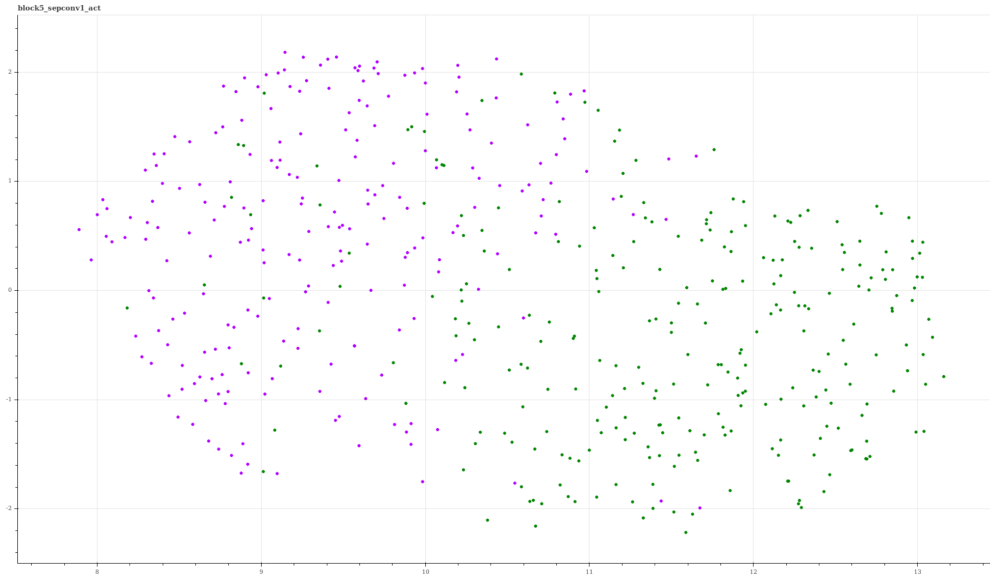

Figure 34. *overcast&lowlight* dataset clustered with the features extracted with *block5_sepconv1_act* layer. AC algorithm with the number of clusters set to 2. NMI score at 0.61.

cab&chicken

The maximum NMI difference in the *cab&chicken* datasets came from layers *block14_sepconv1_bn* and *block14_sep-conv1_act,* when clustering to image categories, regardless of the amount of blur in the blurred images in the dataset. The score difference was 0.93 for the dataset with blur with sigma 2, while 0.96

for the other datasets. The example comparison of the clustering with the layers are shown with the dataset sigma 2 on figure 35. The low NMI score resulted

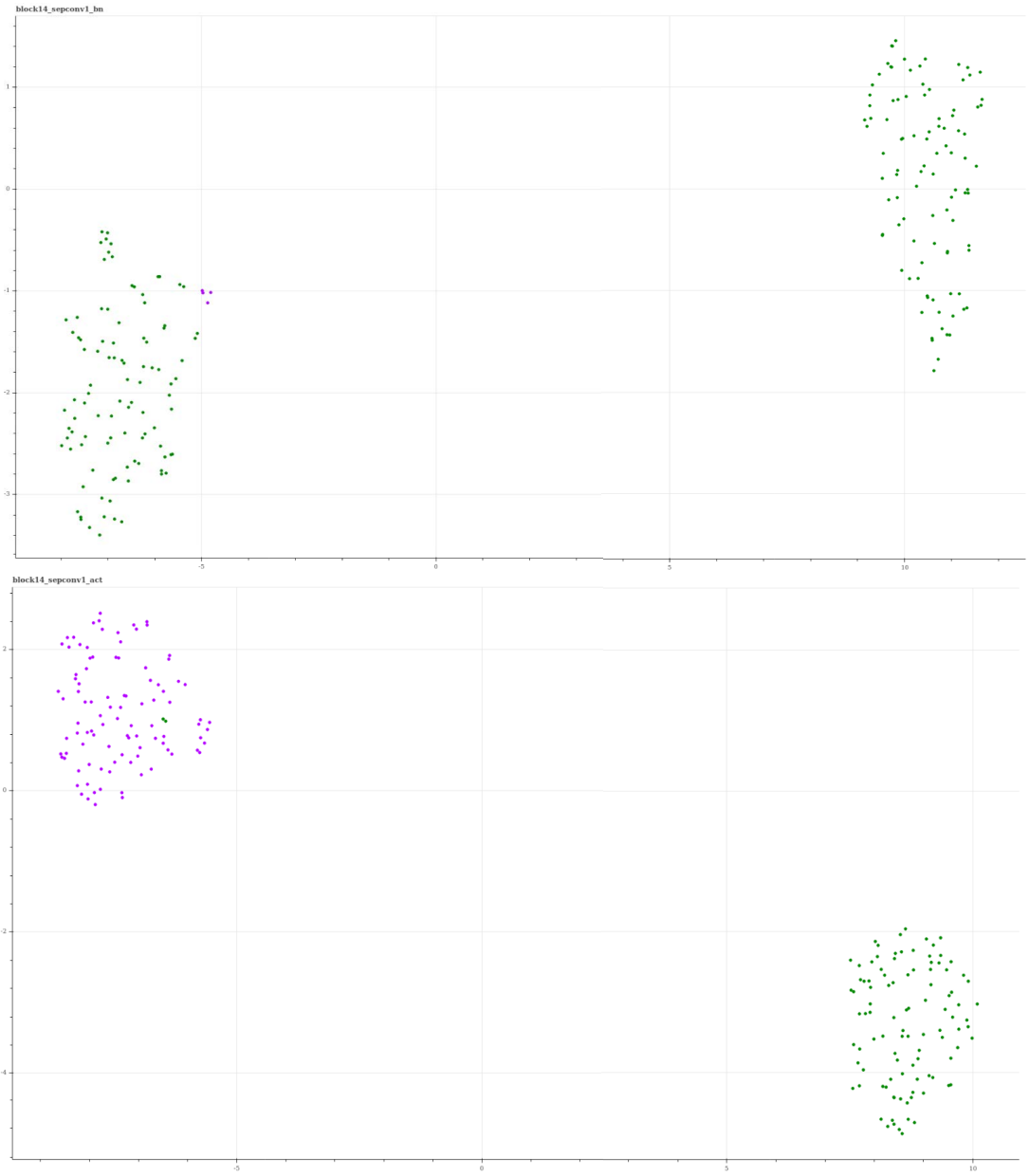

Figure 35. Comparison of *cab&chicken* dataset with blurred images on sigma value 2, clustered image categories with layers *block14_sepconc1_bn* and *block14_sepconv1_act*.

from the unbalanced clustering of the images, as shown in figure 35, where one of the two groups contained only four images while the other had rest of the images (196). In the activation layer *block14_sepconv1_act*, the images were clustered more evenly to both groups. This phenomenon is probably caused by the four outlier images which features are highly different compared to the rest of the images, and the clustering fails similarly as with the *flower&qr* dataset. The same outlier phenomenon can be detected when the features from the layer are reduced

to two dimensions with PCA and clustered with AC. This is illustrated in figure 36, where the four outlier images are the same as when clustered with the high dimensional features. The clustering comparison figures for these layers features with the different sigma valued datasets are shown in appendix 3.

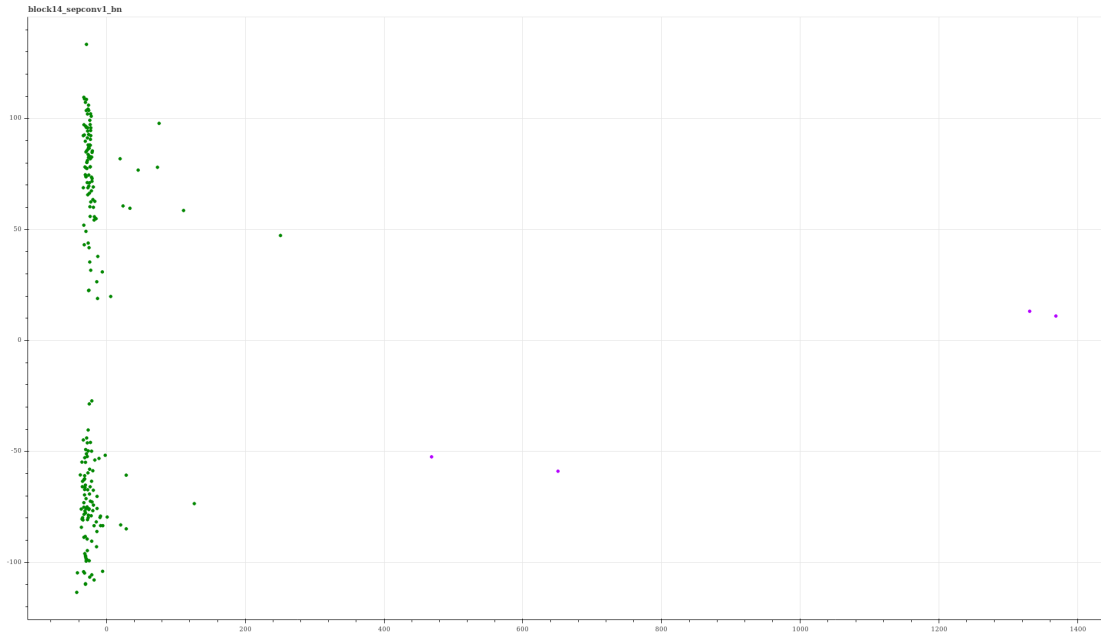

Figure 36. *cab&chicken* dataset with blurred images on sigma value 2 clustered with two-dimensional features extracted with layer *block14_sepconv1_bn* and reduced to two dimensions with PCA.

7. DISCUSSION

The results show that clustering the tested image sets which had more than two categories is best done with the features from the last pooling layer of the CNN (average pool in Xception) before the class prediction layer. Moreover, this was expected behaviour for the CNN because the capability to separate multiple classes from each other increases as deeper layers are selected.

When clustering image sets which had two categories, the selection of which layer to use as feature extractor depended on the image set being clustered. The *flower&qr* dataset could be clustered to correct groups with multiple layers of Xception architecture. Even if the dataset could be clustered perfectly with other than the second last layer, there is probably no benefit in doing so. The guideline for selecting the second last layer as feature extractor should be used when the target domain has similar images than in CNN's training set.

The selection of which layers to extract features with was most impactful when clustering images based on blurriness. The *cab&chicken* datasets results show that some of the layers gave features which could be used to cluster the images based on blurriness, while most of the layers resulted in clustering the two categories to groups. However, to be able to cluster the images based on their blurriness, the images needed to have a substantial amount of blur, as the perfect or near-perfect clusterings came with the images blurred on Gaussian blur with 31×31 kernel size and having sigma value being 9 and 5. The usefulness of detecting a high amount of Gaussian blur in images, which might not correlate to detecting naturally blurry images, is left to be determined.

Clustering is usually conducted to an unknown dataset, and thus selecting the optimal layer to be used as the feature extractor becomes difficult, as the external validation metric for the clustering accuracy can not be used. The most used pipeline for using CNNs as feature extractors in clustering image sets is to use only the second last layer as the feature extractor or fine-tune some of the last layers to the target domain with corresponding training images. However, the usage of other layers as feature extractors need to be studied more, as the results from the *overcast&lowlight* and *cab&chicken* datasets indicate that also other than the few last layers from pre-trained CNNs could be used as feature extractors, depending on the datasets we have and the problem we are solving.

So, should we use pre-trained CNNs and the different layers within for multitude of image feature extraction scenarios in an image set clustering, or is it better to train a specific classification network for each use case separately? For applications like cloud image services or mobile phone galleries, where images from thousands upon thousands of categories can be included, the training of specific classification network which can be used to separate the images, could be challenging. For such applications, this thesis gives an implication that the pre-existing off-the-shelf networks have more capability to extract different image features than the number of classification categories suggests. The different features extracted with the different layers of CNN could be used for generating multiple clusterings of the same image set with one CNN architecture. The images could be clustered to similar groups based by the features extracted from

the upper layers of the CNN architecture and the features from the final layers could be used to fine-tune the clusters to more specific groups.

Future work should include analysis of clustering unlabelled image sets, as to test which kind of images group and how the grouping of the images change on a layer by layer basis. The capability of the layers to separate features from multiple blurred image categories should be tested, as to figure if some of the layers are focused purely on detecting blurriness or does the blur detection depend on the images which were blurred. Another interesting area to study more thoroughly would be, why the neighbouring layer features behaved so differently when clustering them. The qualitative results demonstrated that the selection of which layers to use in feature extraction affects the clustering outcome greatly when neighbouring layers could produce features which cluster very differently.

Overall the evidence gathered here was with a sample of few datasets and on one CNN architecture, which limits the generalisability of the results. For future work, a smaller CNN architecture or architectures could be beneficial, as the more focused study could be conducted with fewer layers and with multiple CNN architectures.

8. CONCLUSION

The objective of this thesis was to test image feature extraction capabilities of a pre-trained CNN's layers for image set clustering. To test which kind of features could be extracted with the layers, eight image sets with different scenarios were selected. While previous studies have focused on using some of the last layers for the feature extraction, the tests in this thesis were conducted with all of the layers in the CNN, to test if the features extracted with different layers could be used for clustering different image sets.

The test results show that the best clustering scores come from different layers depending on the selected image set being clustered. The test results verify the known viable method of selecting some of the last layers for extracting features from images similar to the ones in CNN's training set. However, the test datasets, which focused on blurriness and scene detection showed that the pre-trained CNN's last layers do not always give the best clustering results. On image set clustering applications, where the image feature extractor is selected to be CNN architecture, and the image sets being clustered vary, the optimal layer for the feature extraction should be selected based on the image set. Other findings from the experimental result are, how differently image sets cluster when using adjacent layers from CNN to extract the image features, and thus finding an optimal layer for different image set needs testing.

This thesis suggests that when pre-trained CNNs are used as feature extractors in an image set clustering pipeline, the features should be extracted with multiple layers of the architecture to find the optimal features for the clustering task. However, the selection of the optimal layer has limitations, as the features obtained by a layer used for the feature extraction should be validated with a labelled sample set.

9. REFERENCES

[1] Caron M., Bojanowski P., Joulin A. & Douze M. (2018) Deep clustering for unsupervised learning of visual features. In: Ferrari V., Hebert M., Sminchisescu C., Weiss Y. (eds) Computer Vision - ECCV 2018, ECCV 2018. Lecture Notes in Computer Science, vol 11218. Springer, Cham.

[2] Yosinski J., Clune J., Bengio Y. & Lipson H. (2014) How transferable are features in deep neural networks? In: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence & K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems 27, Curran Associates, Inc., pp. 3320–3328.

[3] Guérin J., Gibaru O., Thiery S. & Nyiri E. (2017) CNN features are also great at unsupervised classification. CoRR abs/1707.01700.

[4] Liu J., Liu J., Du W. & Li D. (2019) Performance Analysis and Characterization of Training Deep Learning Models on NVIDIA TX2. arXiv e-prints , arXiv:1906.04278.

[5] Li D., Chen X., Becchi M. & Zong Z. (2016) Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In: 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), pp. 477–484.

[6] Chollet F. (2017) Xception: Deep learning with depthwise separable convolutions. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1800–1807.

[7] Jain A.K. & Backer E. (1981) A clustering performance measure based on fuzzy set decomposition. IEEE Transactions on Pattern Analysis and Machine Intelligence 3, pp. 66–75.

[8] Hansen P. & Jaumard B. (1997) Cluster analysis and mathematical programming. Mathematical Programming 79, pp. 191–215.

[9] Xu R. & Wunsch D.C. (2009) Clustering. IEEE Series on Computational Intelligence, Wiley-IEEE Press.

[10] Halkidi M., Batistakis Y. & Vazirgiannis M. (2001) On clustering validation techniques. Journal of Intelligent Information Systems 17, pp. 107–145.

[11] Breitkreutz D. & Casey K. (2008) Clusterers: a comparison of partitioning and density-based algorithms and a discussion of optimisations. James Cook University.

[12] Murtagh F. & Legendre P. (2014) Ward's Hierarchical Agglomerative Clustering Method: Which Algorithms Implement Ward's Criterion? Journal of Classification 31, pp. 274–295.

[13] Ester M., Kriegel H.P., Sander J. & Xu X. (1996) A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD'96, AAAI Press, pp. 226–231.

[14] Campello R.J.G.B., Moulavi D. & Sander J. (2013) Density-based clustering based on hierarchical density estimates. In: J. Pei, V.S. Tseng, L. Cao, H. Motoda & G. Xu (eds.) Advances in Knowledge Discovery and Data Mining, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 160–172.

[15] McInnes L., Healy J. & Astels S. (2017) hdbscan: Hierarchical density based clustering. The Journal of Open Source Software 2, p. 205.

[16] Dijkstra E.W. (1959) A note on two problems in connexion with graphs. NUMERISCHE MATHEMATIK 1, pp. 269–271.

[17] de Berg M., Gunawan A. & Roeloffzen M. (2017) Faster db-scan and hdb-scan in low-dimensional euclidean spaces. CoRR abs/1702.08607.

[18] Liu Y., Li Z., Xiong H., Gao X. & Wu J. (2010) Understanding of internal clustering validation measures. In: 2010 IEEE International Conference on Data Mining, pp. 911–916.

[19] Rousseeuw P. (1987) Rousseeuw, p.j.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. comput. appl. math. 20, 53-65. Journal of Computational and Applied Mathematics 20, pp. 53–65.

[20] Rand W.M. (1971) Objective criteria for the evaluation of clustering methods. Journal of the American Statistical Association 66, pp. 846–850.

[21] Vinh N.X., Epps J. & Bailey J. (2010) Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. J. Mach. Learn. Res. 11, pp. 2837–2854.

[22] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M. & Duchesnay E. (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, pp. 2825–2830.

[23] Goodfellow I., Bengio Y. & Courville A. (2016) Deep Learning. MIT Press. `http://www.deeplearningbook.org`.

[24] Hubel D.H. & Wiesel T. (1962) Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology 160, pp. 106–154.

[25] Fukushima K. (1980) Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics 36, pp. 193–202.

[26] LeCun Y., Bottou L., Bengio Y. & Haffner P. (1998) Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, pp. 2278–2324.

[27] LeCun Y., Bengio Y. & Hinton G. (2015) Deep learning. Nature 521, pp. 436–444.

[28] Lai E. (2004) Practical Digital Signal Processing. Newnes.

[29] Zhang R. (2019) Making Convolutional Networks Shift-Invariant Again. arXiv e-prints , arXiv:1904.11486.

[30] Wiki C.S. (2018), Max-pooling / pooling — computer science wiki,. URL: https://computersciencewiki.org/index.php?title=Max-pooling_/_Pooling&oldid=7839, [Online; accessed 8-April-2019].

[31] He K., Zhang X., Ren S. & Sun J. (2016) Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778.

[32] Ioffe S. & Szegedy C. (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: 32nd International Conference on Machine Learning, ICML 2015, pp. 448–456.

[33] Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V. & Rabinovich A. (2015) Going deeper with convolutions. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 1–9.

[34] Sifre L. & Mallat S. (2014) Rigid-motion scattering for image classification. Ph. D. dissertation .

[35] Santurkar S., Tsipras D., Ilyas A. & Madry A. (2018) How Does Batch Normalization Help Optimization? arXiv e-prints , arXiv:1805.11604.

[36] Kohler J., Daneshmand H., Lucchi A., Zhou M., Neymeyr K. & Hofmann T. (2018) Exponential convergence rates for Batch Normalization: The power of length-direction decoupling in non-convex optimization. arXiv e-prints , arXiv:1805.10694.

[37] Razavian A.S., Azizpour H., Sullivan J. & Carlsson S. (2014) CNN Features Off-the-Shelf: An Astounding Baseline for Recognition. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 512–519.

[38] Krasin I., Duerig T., Alldrin N., Ferrari V., Abu-El-Haija S., Kuznetsova A., Rom H., Uijlings J., Popov S., Veit A. et al. (2017) Openimages: A public dataset for large-scale multi-label and multi-class image classification. Dataset available from https://github. com/openimages 2, p. 3.

[39] Deng J., Dong W., Socher R., Li L.J., Li K. & Fei-Fei L. (2009) ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09.

[40] Everingham M., Van Gool L., Williams C.K.I., Winn J. & Zisserman A. (2010) The pascal visual object classes (voc) challenge. International Journal of Computer Vision 88, pp. 303–338.

[41] Krizhevsky A., Hinton G. et al. (2009) Learning multiple layers of features from tiny images. Tech. rep., University of Toronto, Toronto.

[42] Russakovsky O., Deng J., Su H., Krause J., Satheesh S., Ma S., Huang Z., Karpathy A., Khosla A., Bernstein M., Berg A.C. & Fei-Fei L. (2015) Imagenet large scale visual recognition challenge. International Journal of Computer Vision 115, pp. 211–252.

[43] Tan M. & Le Q.V. (2019) EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv e-prints , arXiv:1905.11946.

[44] Bellman R. (1957) Dynamic Programming. Princeton University Press, Princeton, NJ, USA, 1 ed.

[45] Kuo F.Y. & Sloan I.H. (2005) Lifting the curse of dimensionality. Notices of the AMS 52, pp. 1320–1329.

[46] Pearson K. (1901) Liii. on lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2, pp. 559–572.

[47] Hyvärinen A., Karhunen J. & Oja E. (2004) Independent component analysis, vol. 46. John Wiley & Sons.

[48] McInnes L., Healy J. & Melville J. (2018) UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. ArXiv e-prints .

[49] Roweis S.T. & Saul L.K. (2000) Nonlinear dimensionality reduction by locally linear embedding. Science 290, pp. 2323–2326.

[50] Hinton G.E. & Salakhutdinov R.R. (2006) Reducing the dimensionality of data with neural networks. Science 313, pp. 504–507.

[51] Sorzano C.O.S., Vargas J. & Montano A.P. (2014) A survey of dimensionality reduction techniques. arXiv e-prints , arXiv:1403.2877.

[52] McInnes L., Healy J., Saul N. & Grossberger L. (2018) Umap: Uniform manifold approximation and projection. The Journal of Open Source Software 3, p. 861.

[53] Abadi M., Agarwal A., Barham P., Brevdo E., Chen Z., Citro C., Corrado G.S., Davis A., Dean J., Devin M., Ghemawat S., Goodfellow I., Harp A., Irving G., Isard M., Jia Y., Jozefowicz R., Kaiser L., Kudlur M., Levenberg J., Mané D., Monga R., Moore S., Murray D., Olah C., Schuster M., Shlens J., Steiner B., Sutskever I., Talwar K., Tucker P., Vanhoucke V., Vasudevan V., Viégas F., Vinyals O., Warden P., Wattenberg M., Wicke M., Yu Y. & Zheng X. (2015), TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[54] Chollet F. et al. (2015), Keras. `https://keras.io`.

10. APPENDICES

Appendix 1. Table comparing the maximum NMI scores of dimension reduced and native feature clusters with HDBSCAN.

Appendix 2. Figures of NMI scores for the evaluation datasets with UMAP and PCA dimension reduced features also results with the HDBSCAN clustering algorithm are included.

Appendix 3. Clustering plot comparisons of layers with a maximum difference in NMI scores with the *cab&chicken* datasets

Table 1. Highest NMI scores of HDBSCAN clustering

Datasets	Best NMI score with DR	Best NMI score without DR
ImageNet10	(UMAP) 0.95	0.76
flower&qr	(UMAP) 1.0	0.62
Tools	(UMAP) 0.67	0.47
overcast&lowlight	(PCA) 0.55	0.24
sigma2(Category)	(UMAP) 1.0	0.79
sigma3(Category)	(UMAP) 1.0	0.70
sigma5(Category)	(UMAP) 1.0	0.65
sigma9(Category)	(UMAP) 1.0	0.61
sigma2(Blur)	(PCA) 0.64	0.10
sigma3(Blur)	(PCA) 0.70	0.10
sigma5(Blur)	(PCA) 0.92	0.41
sigma9(Blur)	(PCA) 0.90	0.83

HDBSCAN clustering accuracies increase with all of the datasets when using dimension reduction.

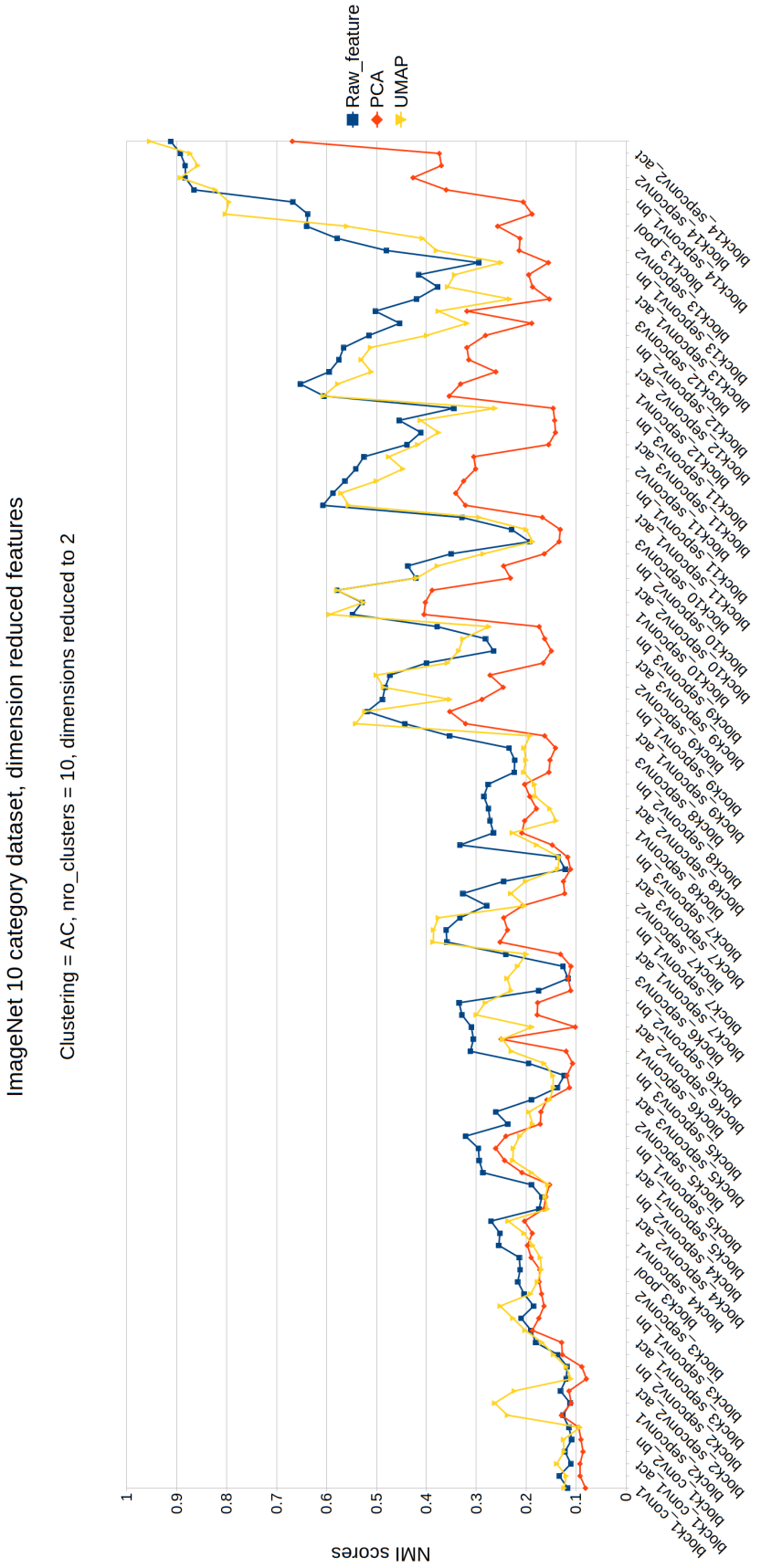

Figure 1. ImageNet 10 category dataset features clustered with original and PCA and UMAP reduced dimensions. Clustering with AC.

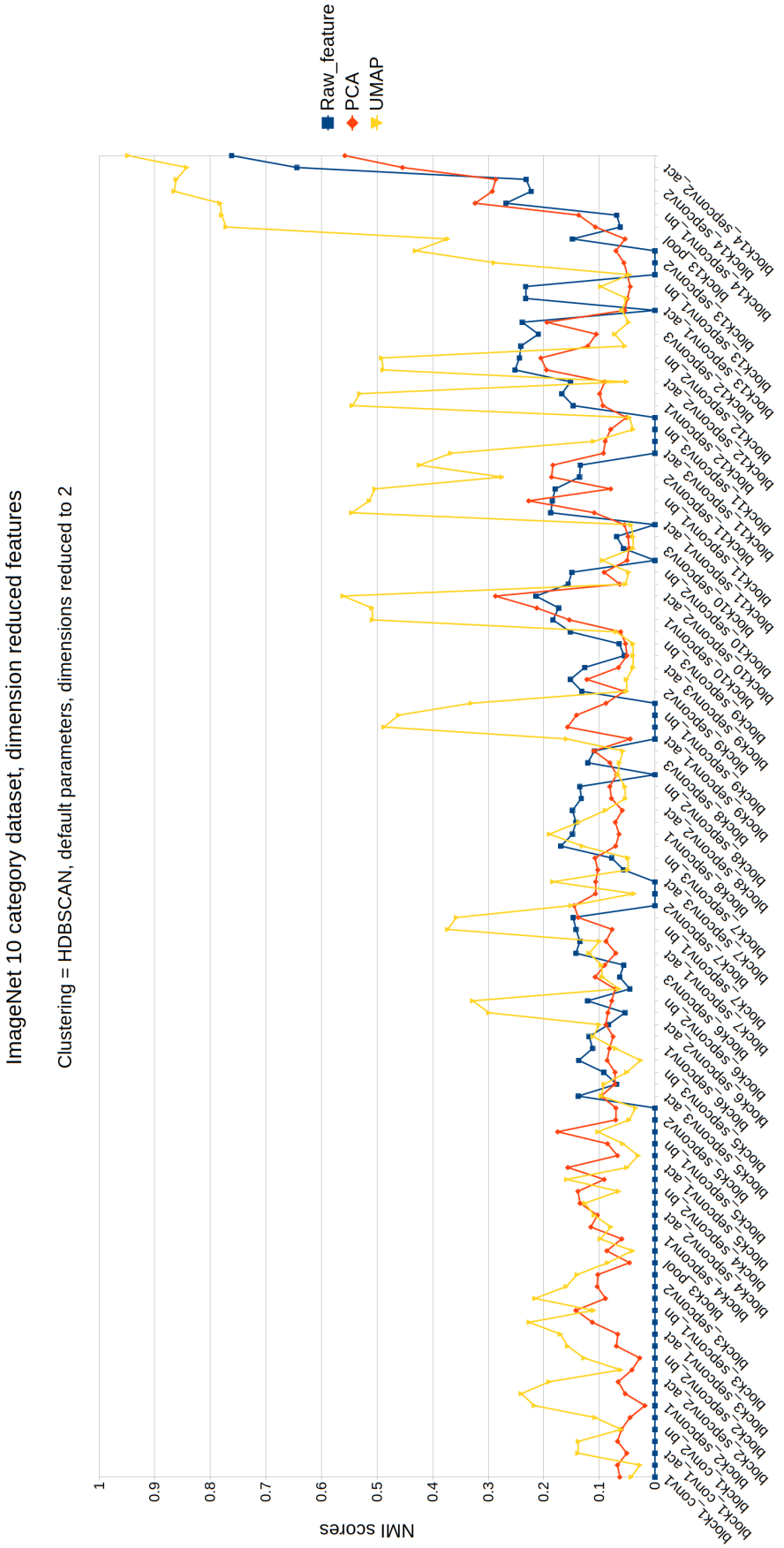

Figure 2. ImageNet 10 category dataset features clustered with original and PCA and UMAP reduced dimensions. Clustering with HDBSCAN.

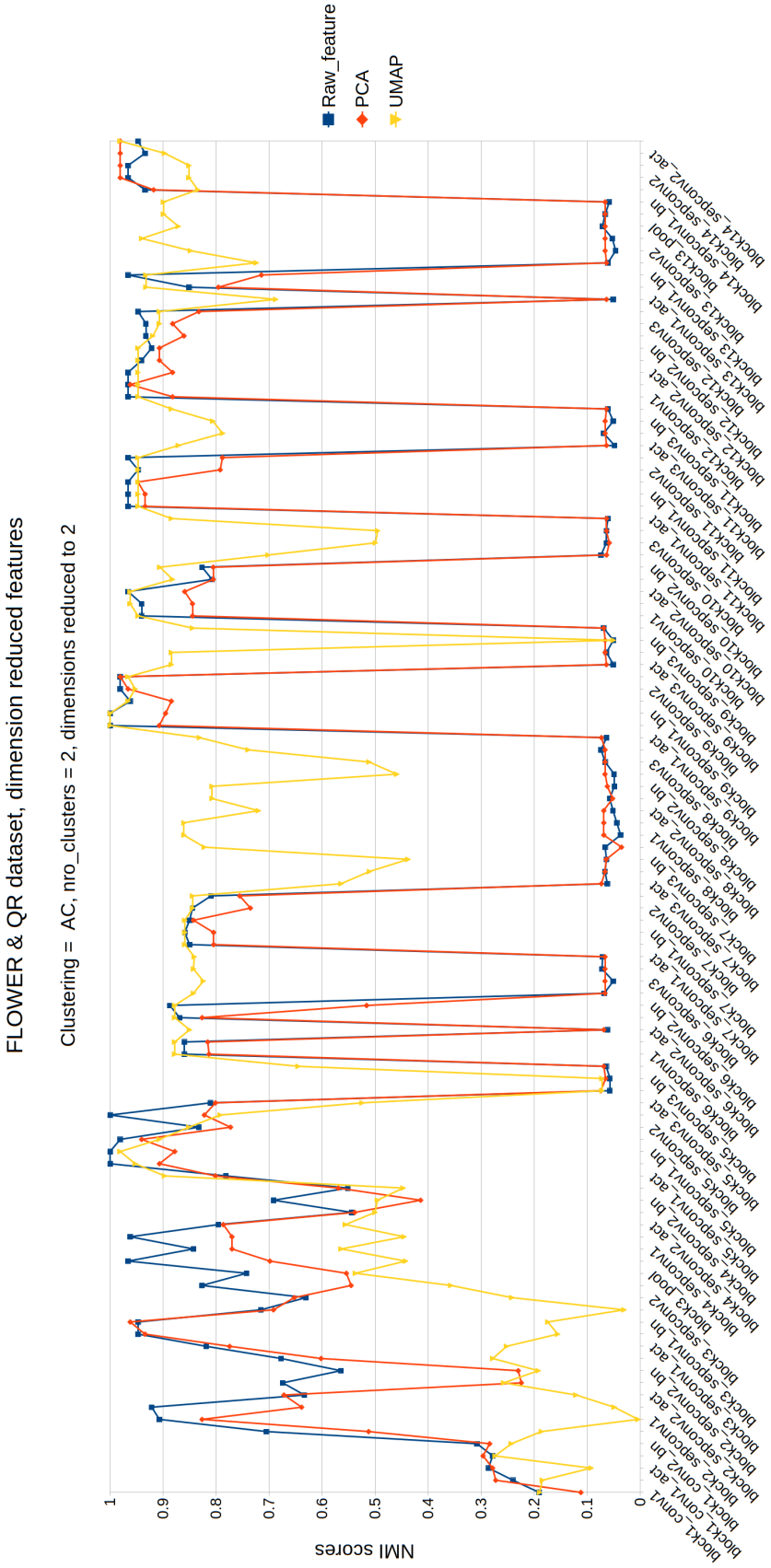

Figure 3. FLOWER & QR dataset features clustered with original and PCA and UMAP reduced dimensions. Clustering with AC.

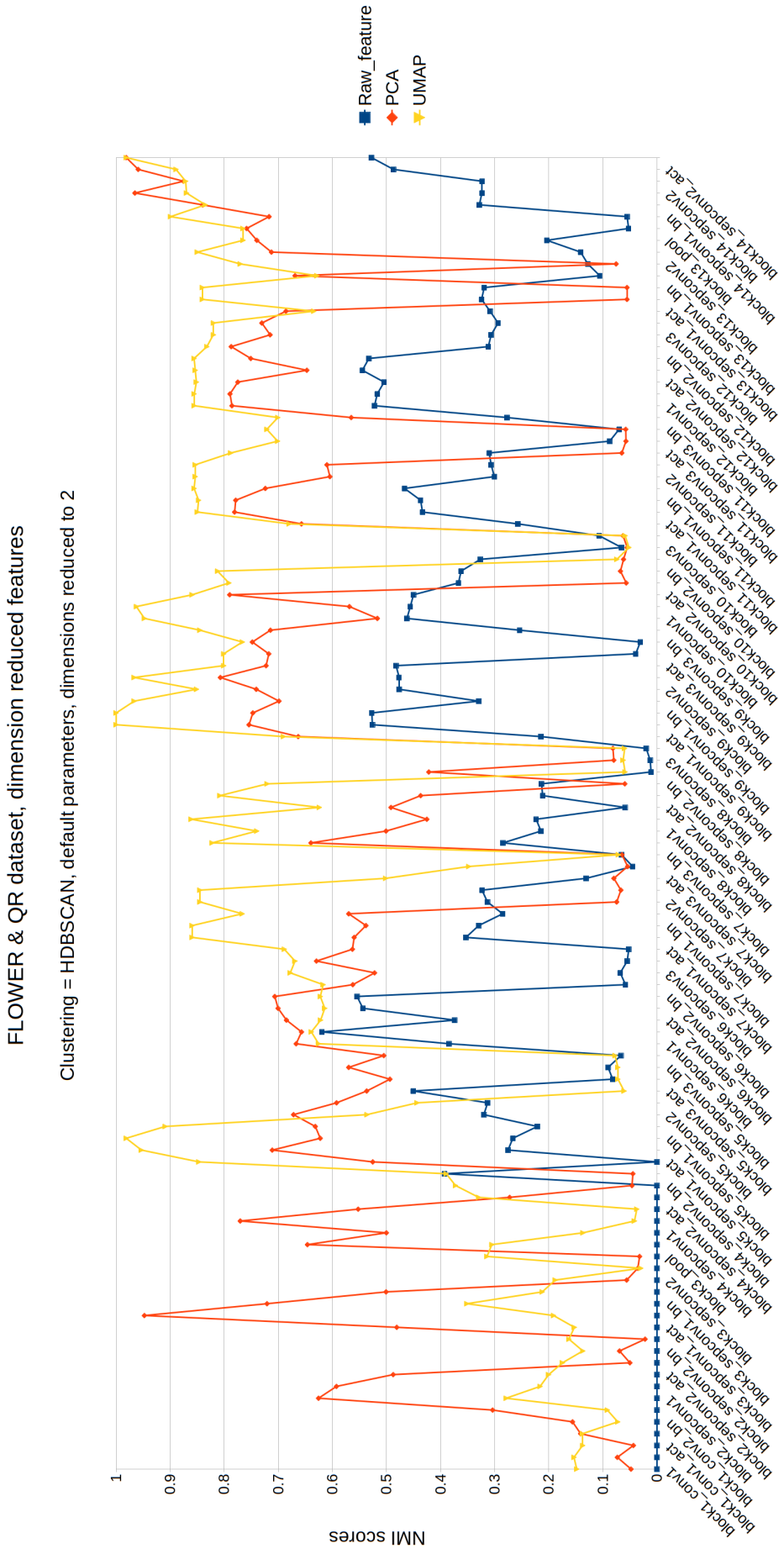

Figure 4. FLOWER & QR dataset features clustered with original and PCA and UMAP reduced dimensions. Clustering with HDBSCAN.

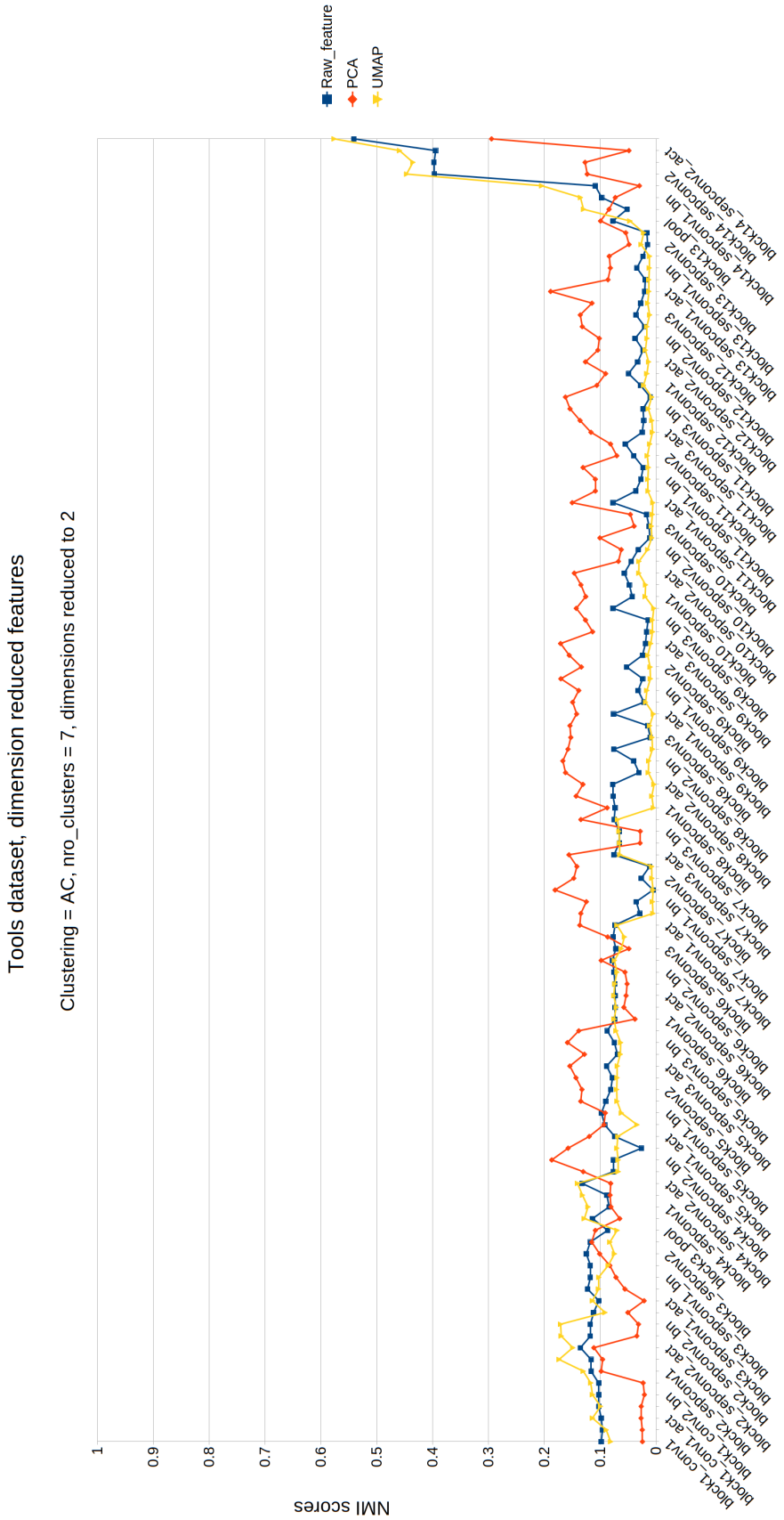

Figure 5. Tools dataset features clustered with original and PCA and UMAP reduced dimensions. Clustering with AC.

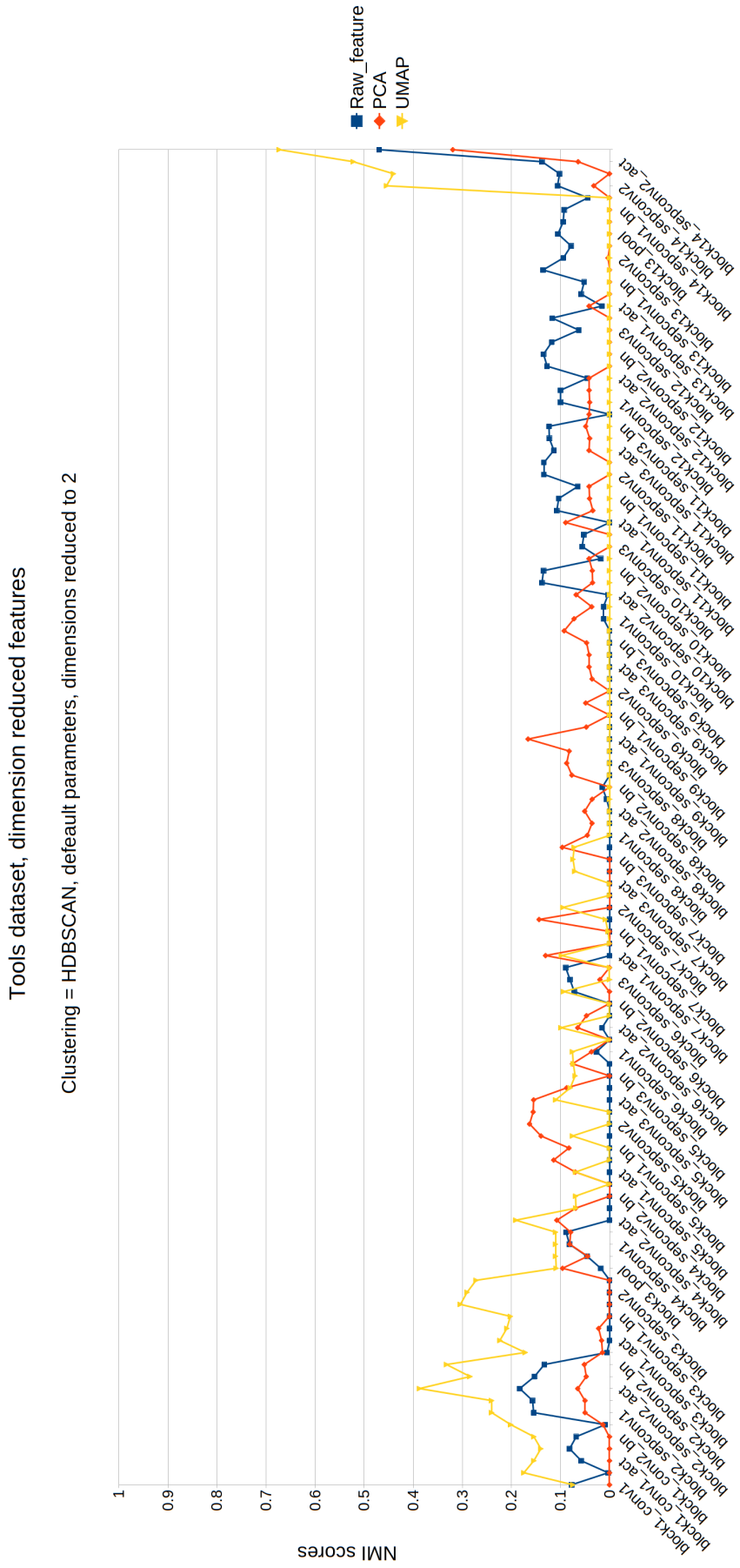

Figure 6. Tools dataset features clustered with original and PCA and UMAP reduced dimensions. Clustering with HDBSCAN.

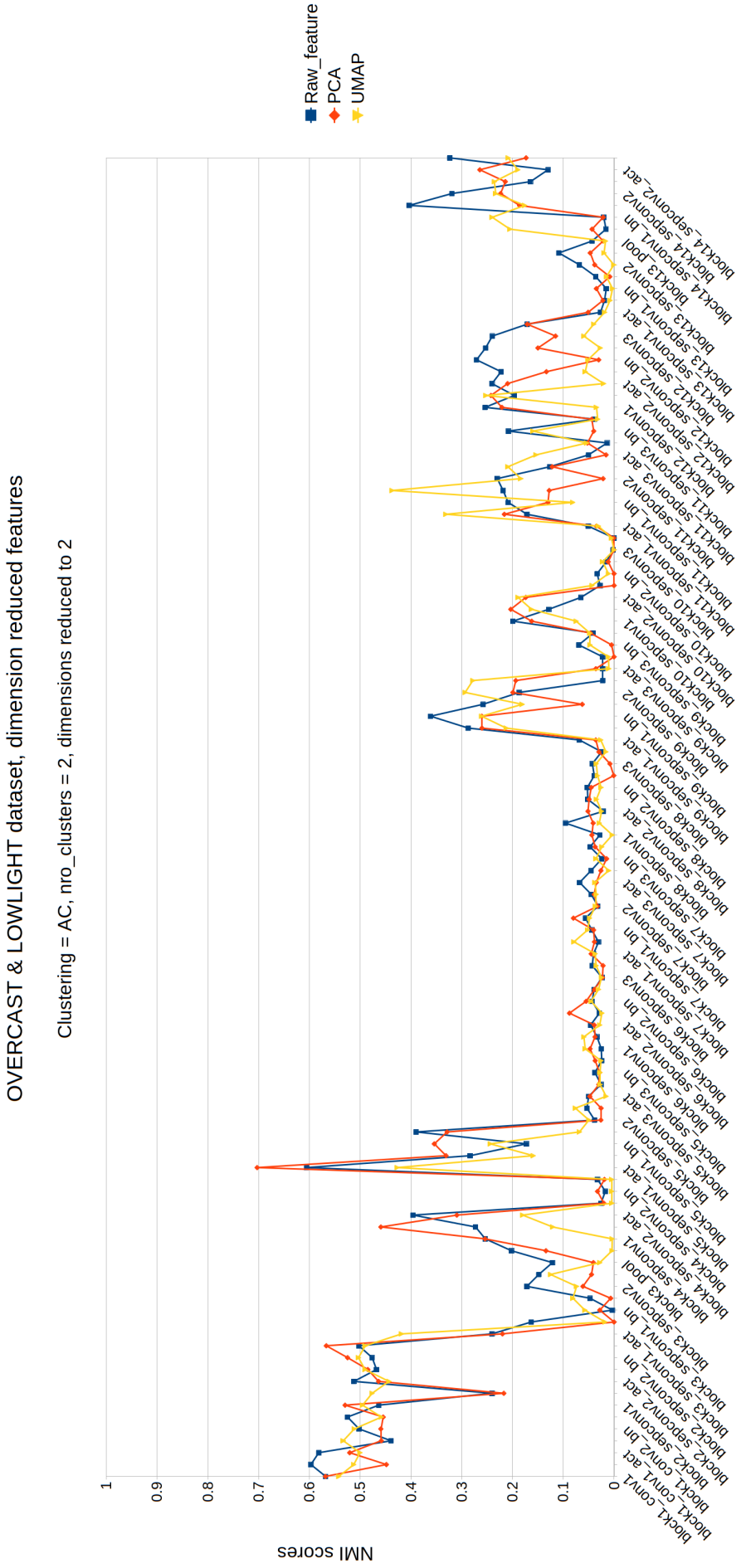

Figure 7. OVERCAST & LOWLIGHT dataset features clustered with original and PCA and UMAP reduced dimensions. Clustering with AC.

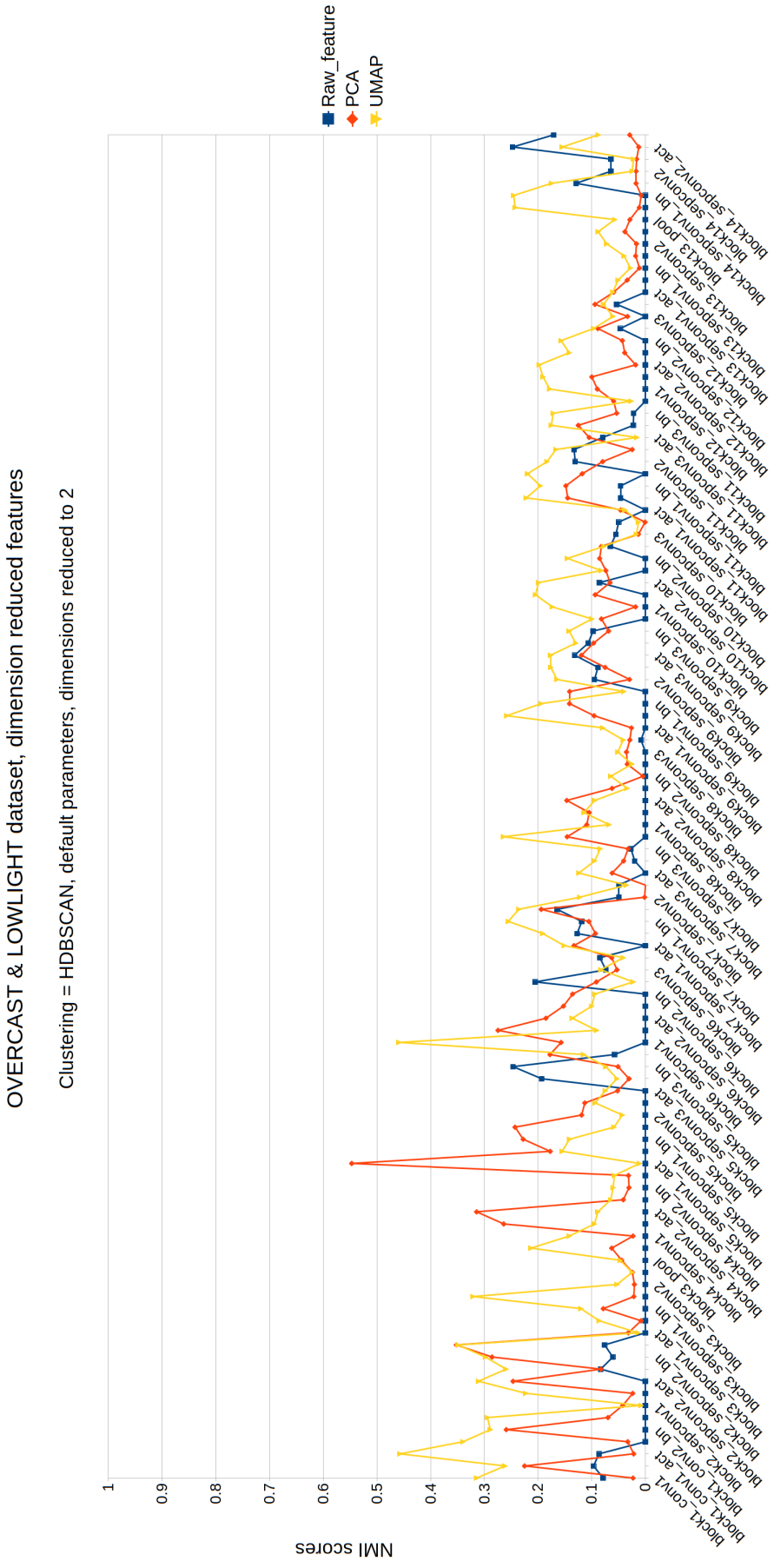

Figure 8. OVERCAST & LOWLIGHT dataset features clustered with original and PCA and UMAP reduced dimensions. Clustering with HDBSCAN.

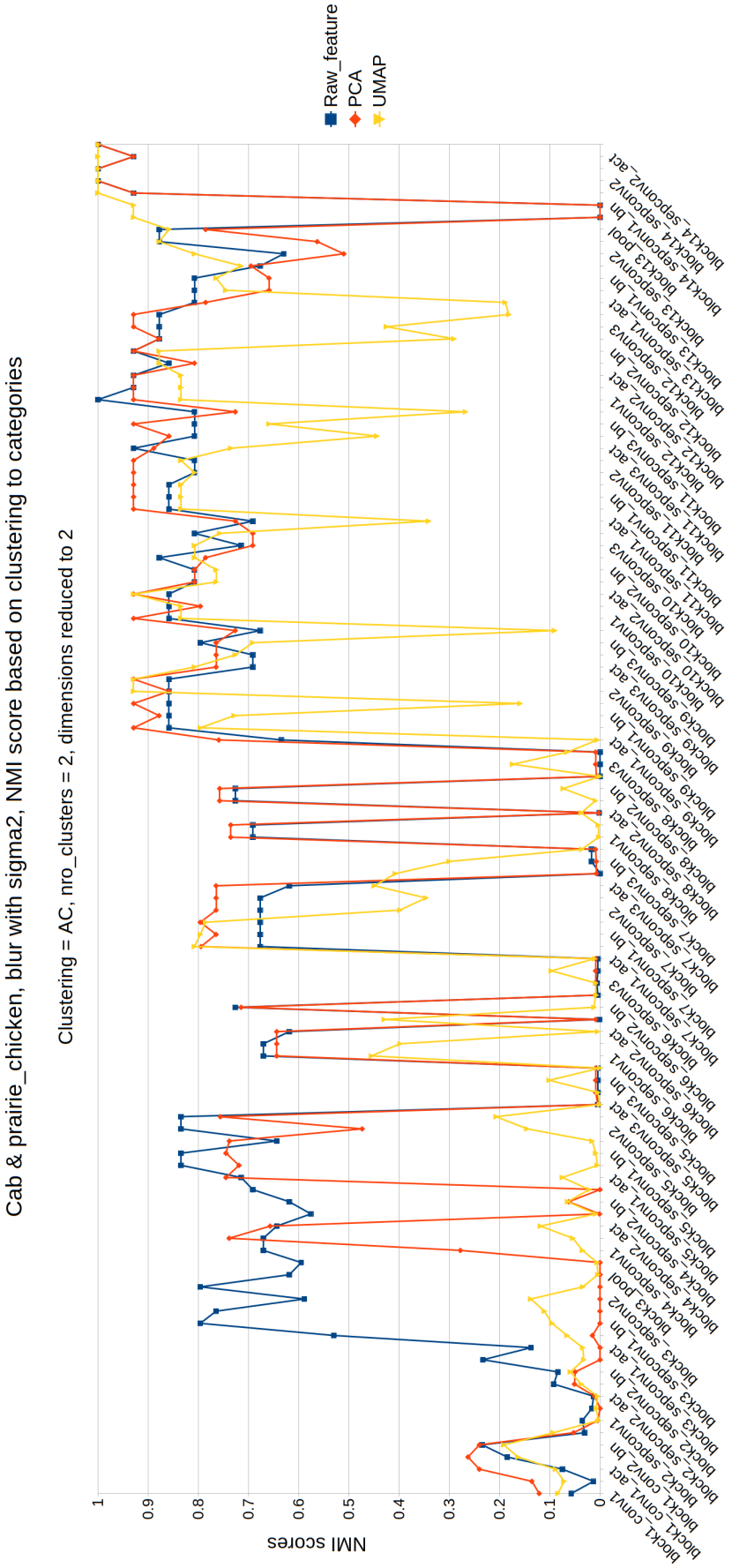

Figure 9. Cab & prairie_chicken with sigma2 blur dataset features clustered to categories with original and PCA and UMAP reduced dimensions. Clustering with AC.

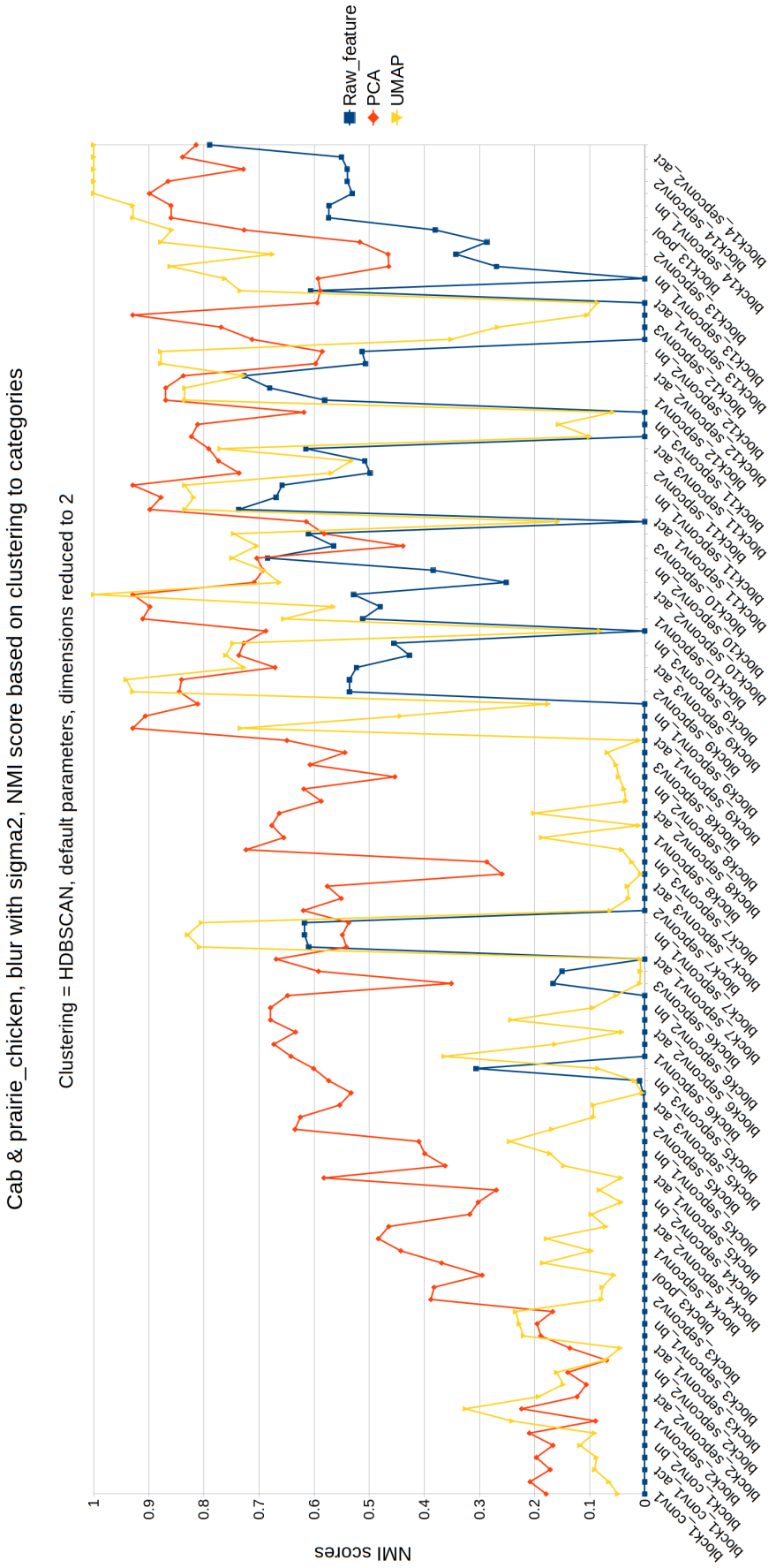

Figure 10. Cab & prairie_chicken with sigma2 blur dataset features clustered to categories with original and PCA and UMAP reduced dimensions. Clustering with HDBSCAN.

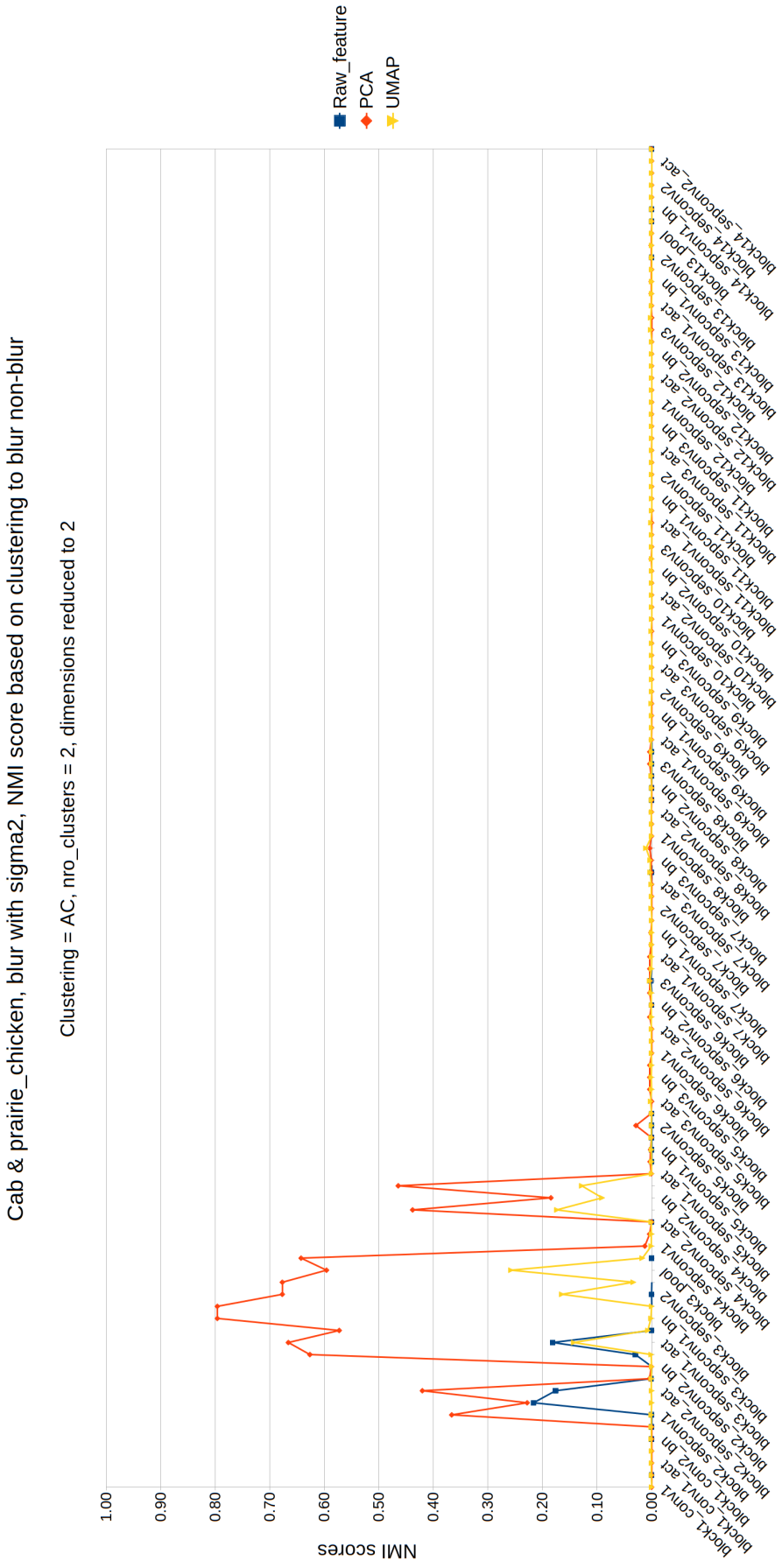

Figure 11. Cab & prairie_chicken with sigma2 blur dataset features clustered to blur and non-blur groups with original and PCA and UMAP reduced dimensions. Clustering with AC.

Figure 12. Cab & prairie_chicken with sigma2 blur dataset features clustered to blur and non-blur groups with original and PCA and UMAP reduced dimensions. Clustering with HDBSCAN.

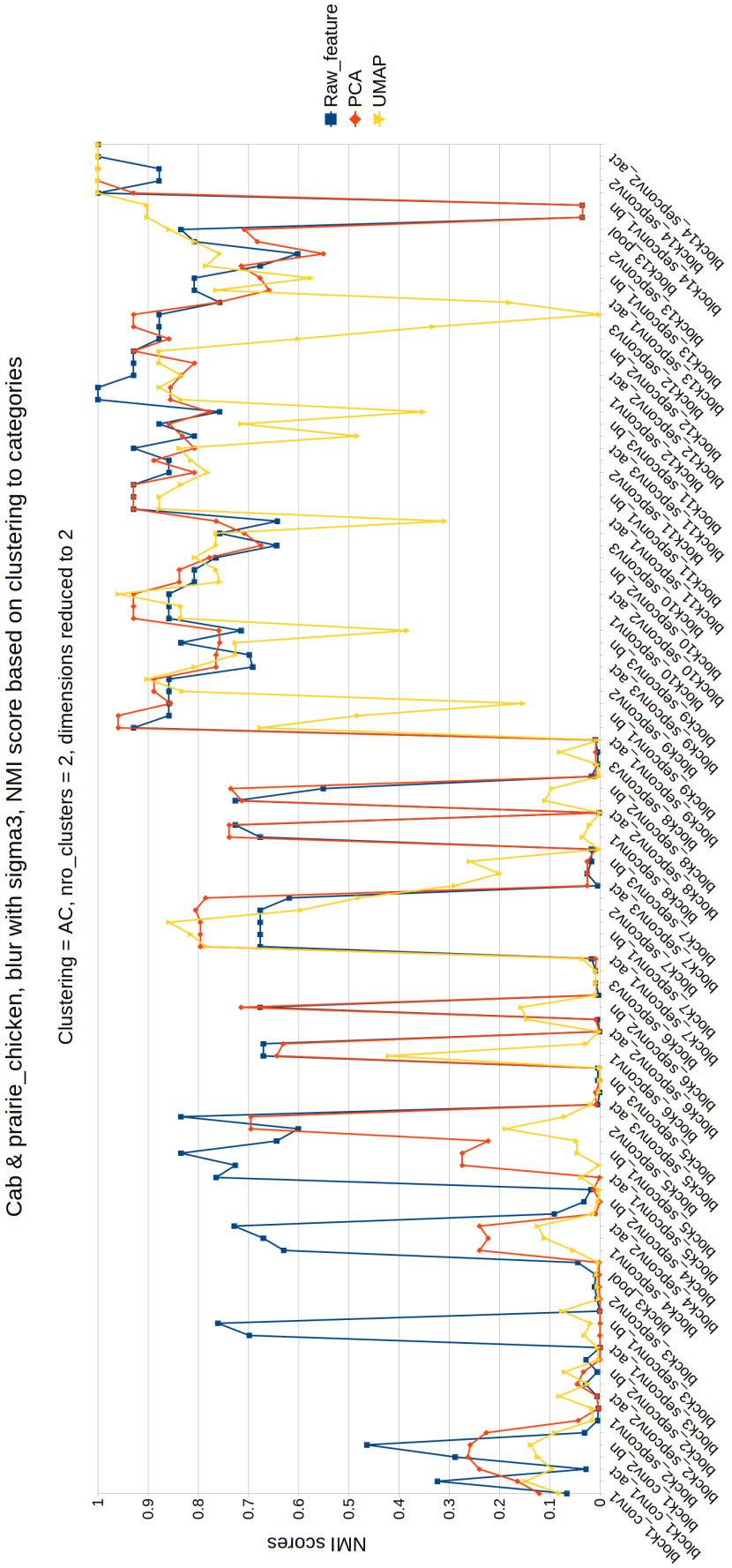

Figure 13. Cab & prairie_chicken with sigma3 blur dataset features clustered to categories with original and PCA and UMAP reduced dimensions. Clustering with AC.

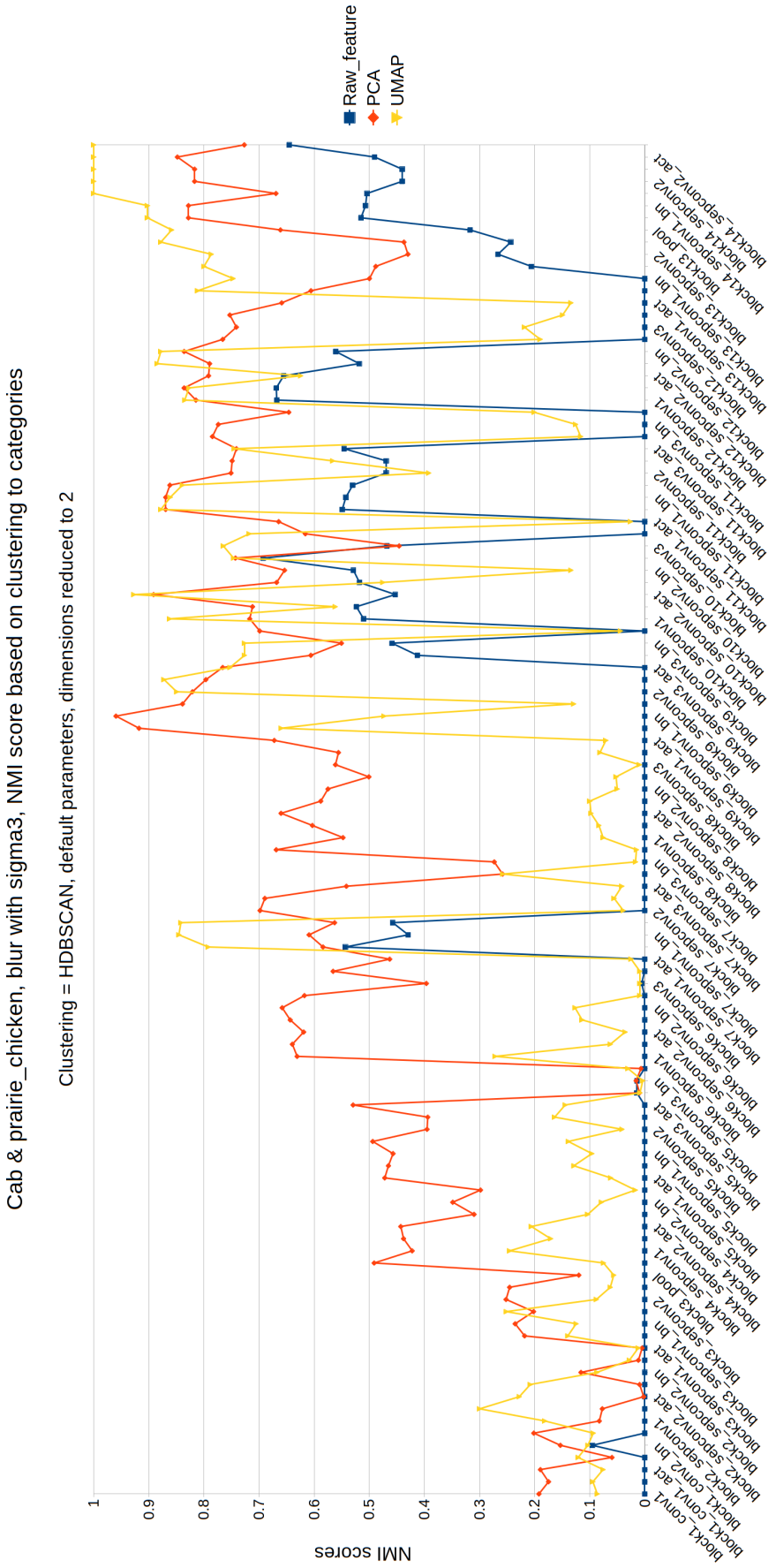

Figure 14. Cab & prairie_chicken with sigma3 blur dataset features clustered to categories with original and PCA and UMAP reduced dimensions. Clustering with HDBSCAN.

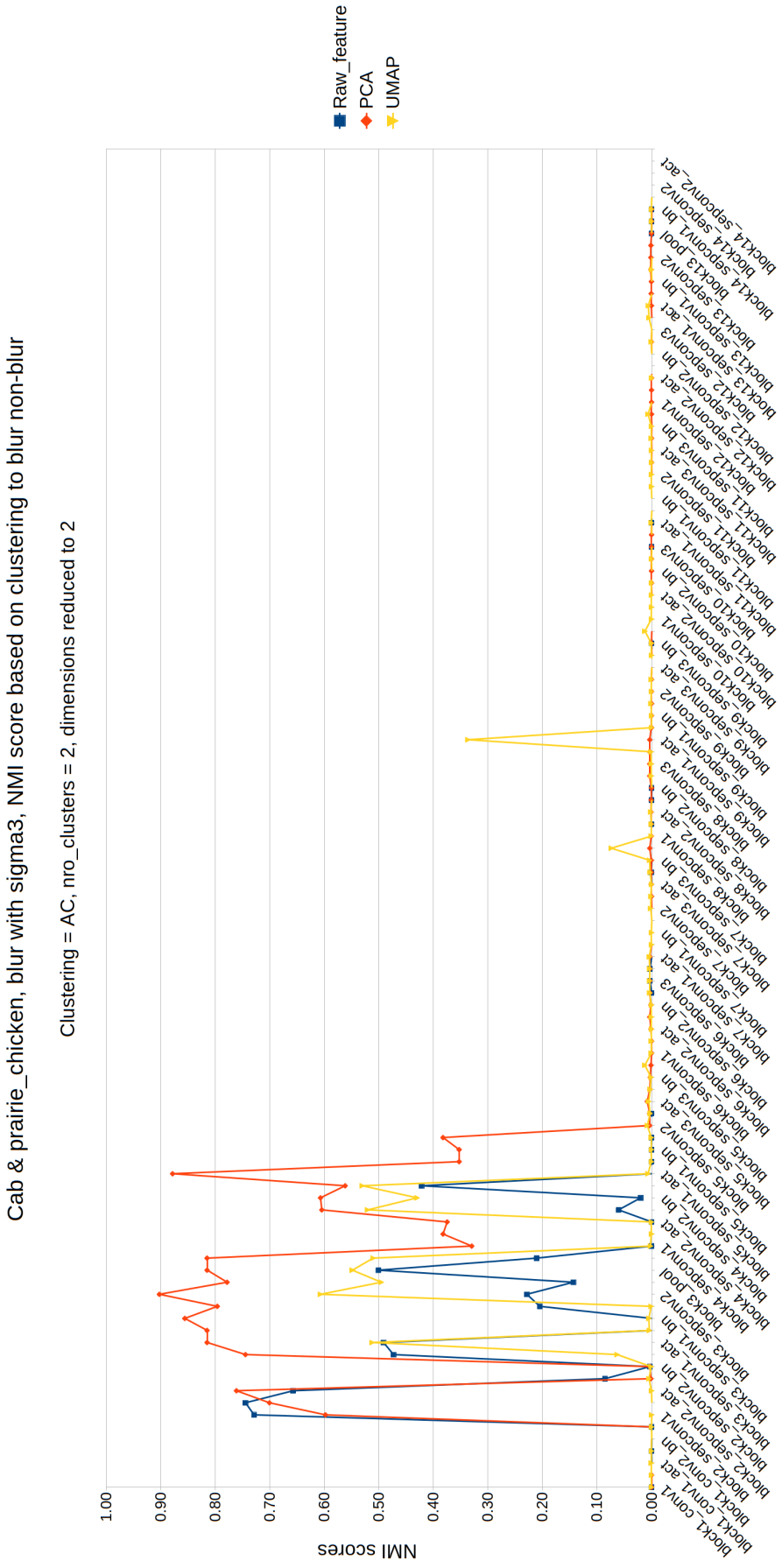

Figure 15. Cab & prairie_chicken with sigma3 blur dataset features clustered to blur and non-blur groups with original and PCA and UMAP reduced dimensions. Clustering with AC.

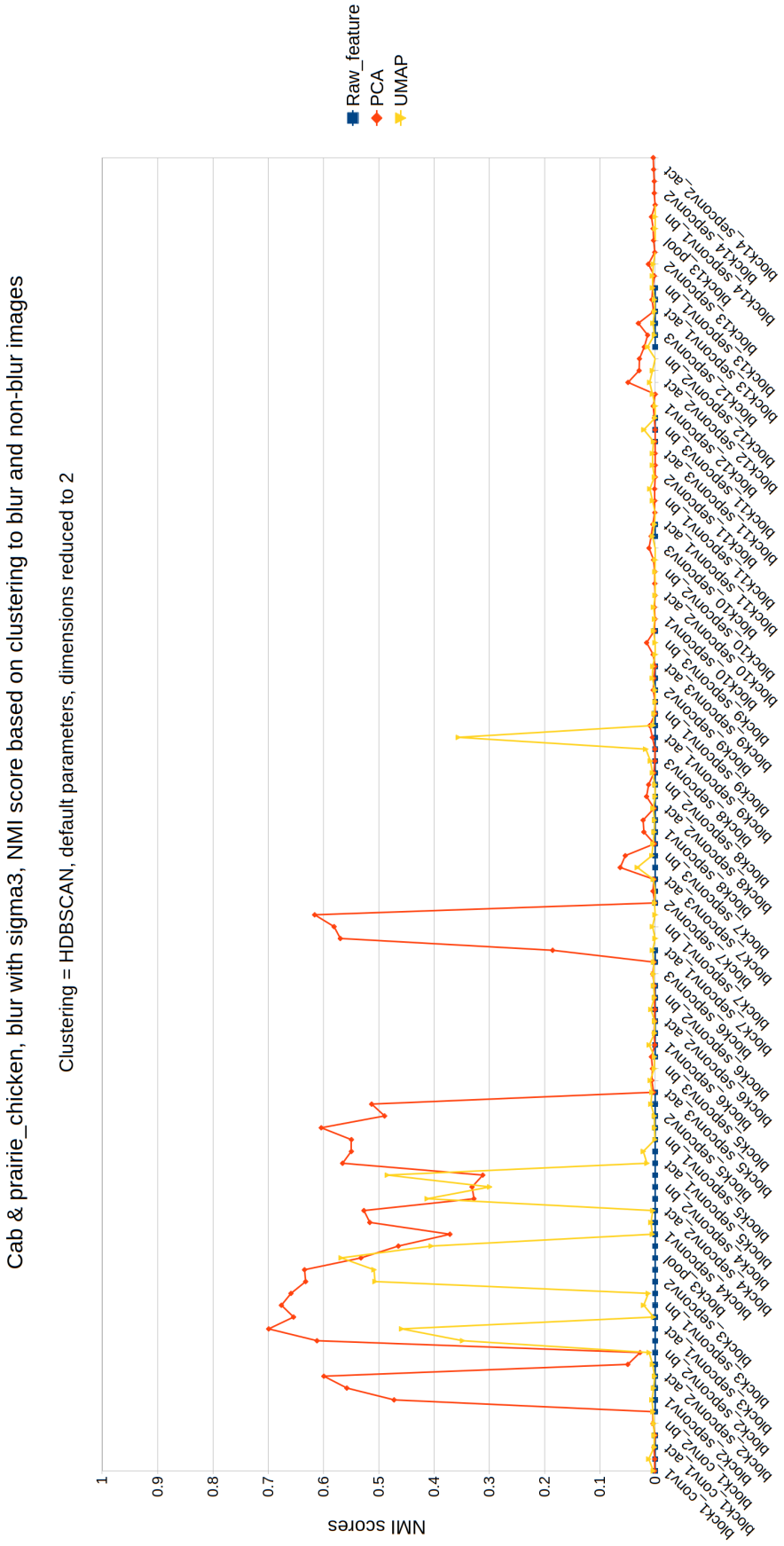

Figure 16. Cab & prairie_chicken with sigma3 blur dataset features clustered to blur and non-blur groups with original and PCA and UMAP reduced dimensions. Clustering with HDBSCAN.

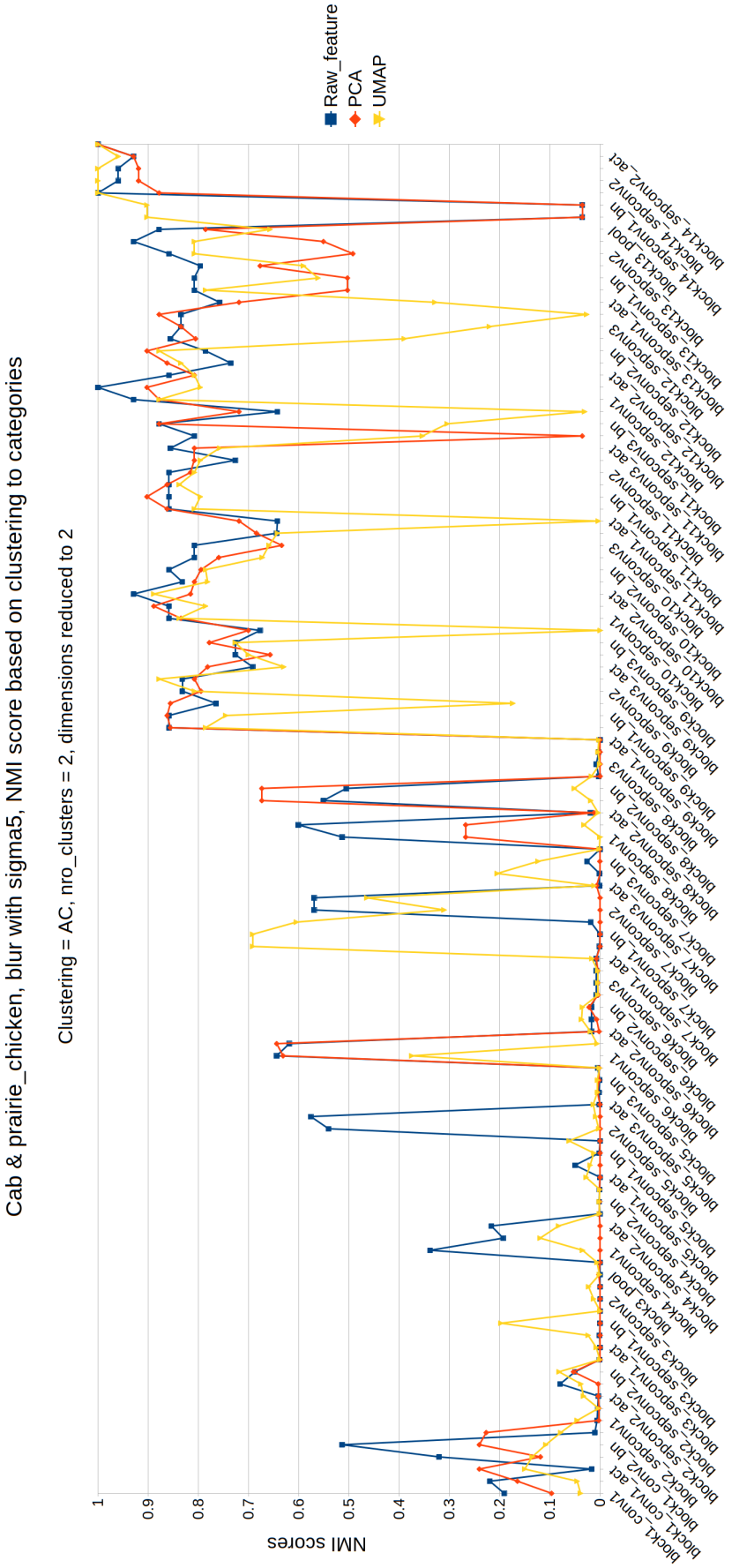

Figure 17. Cab & prairie_chicken with sigma5 blur dataset features clustered to categories with original and PCA and UMAP reduced dimensions. Clustering with AC.

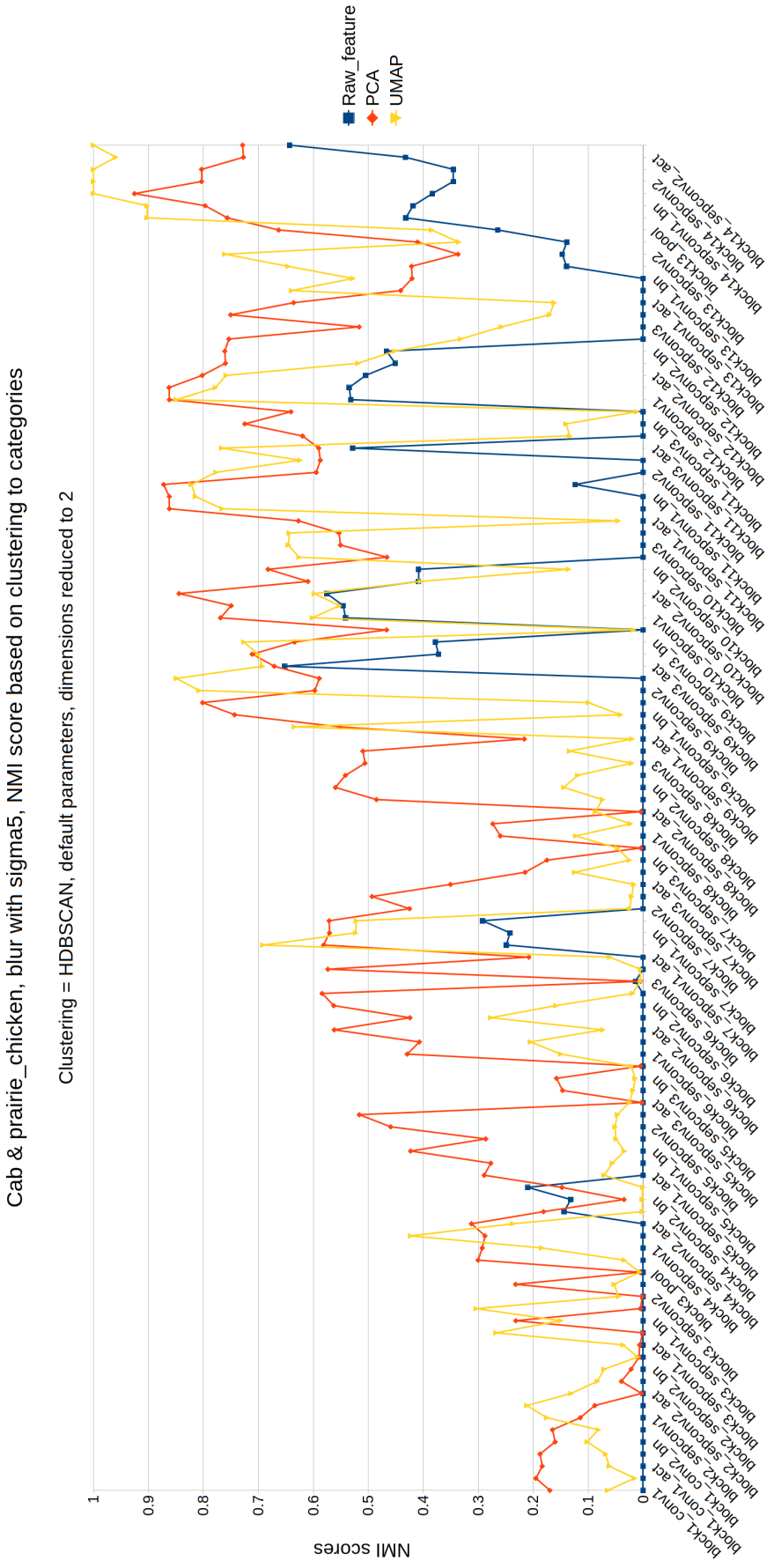

Figure 18. Cab & prairie_chicken with sigma5 blur dataset features clustered to categories with original and PCA and UMAP reduced dimensions. Clustering with HDBSCAN.

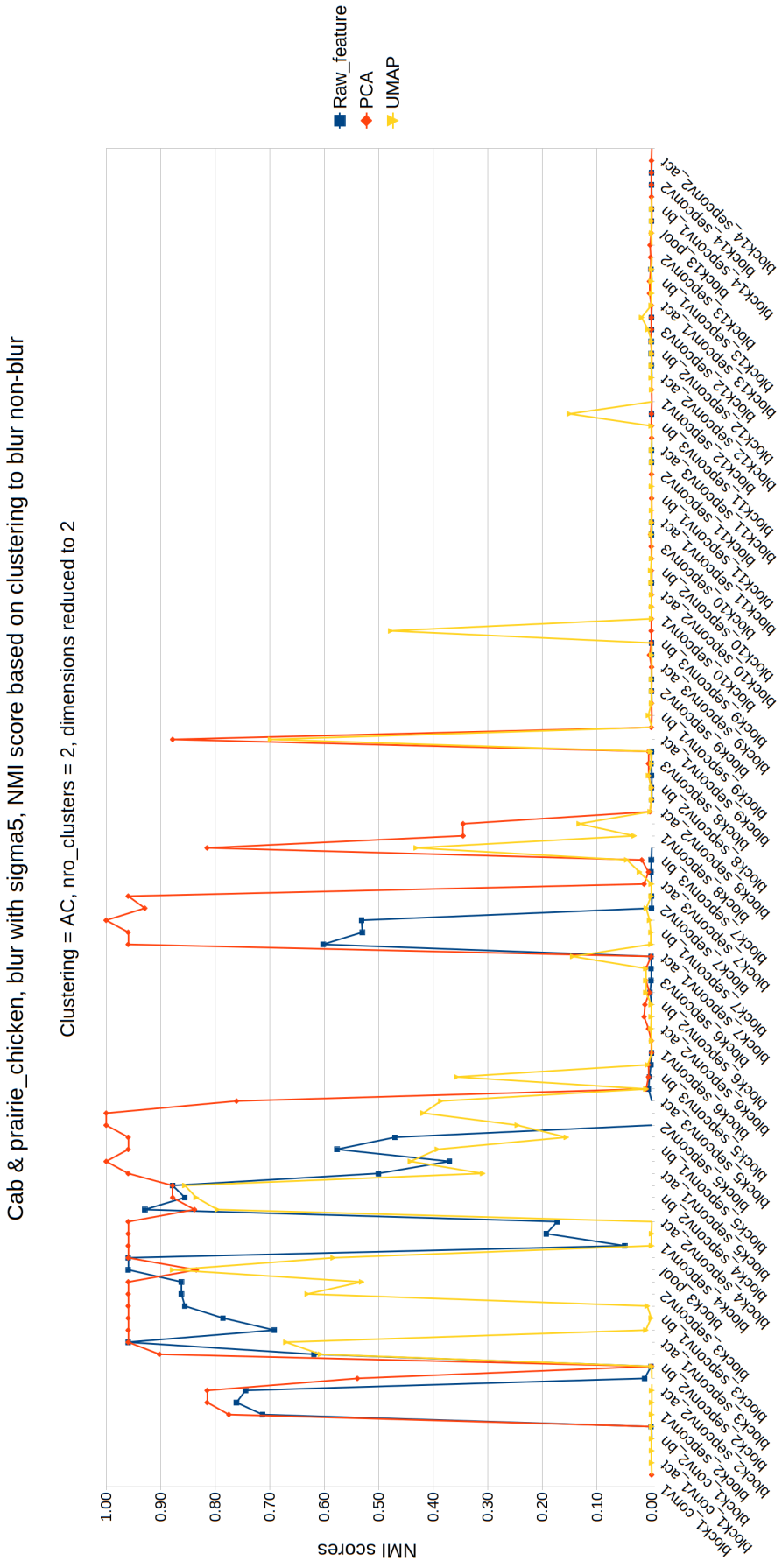

Figure 19. Cab & prairie_chicken with sigma5 blur dataset features clustered to blur and non-blur groups with original and PCA and UMAP reduced dimensions. Clustering with AC.

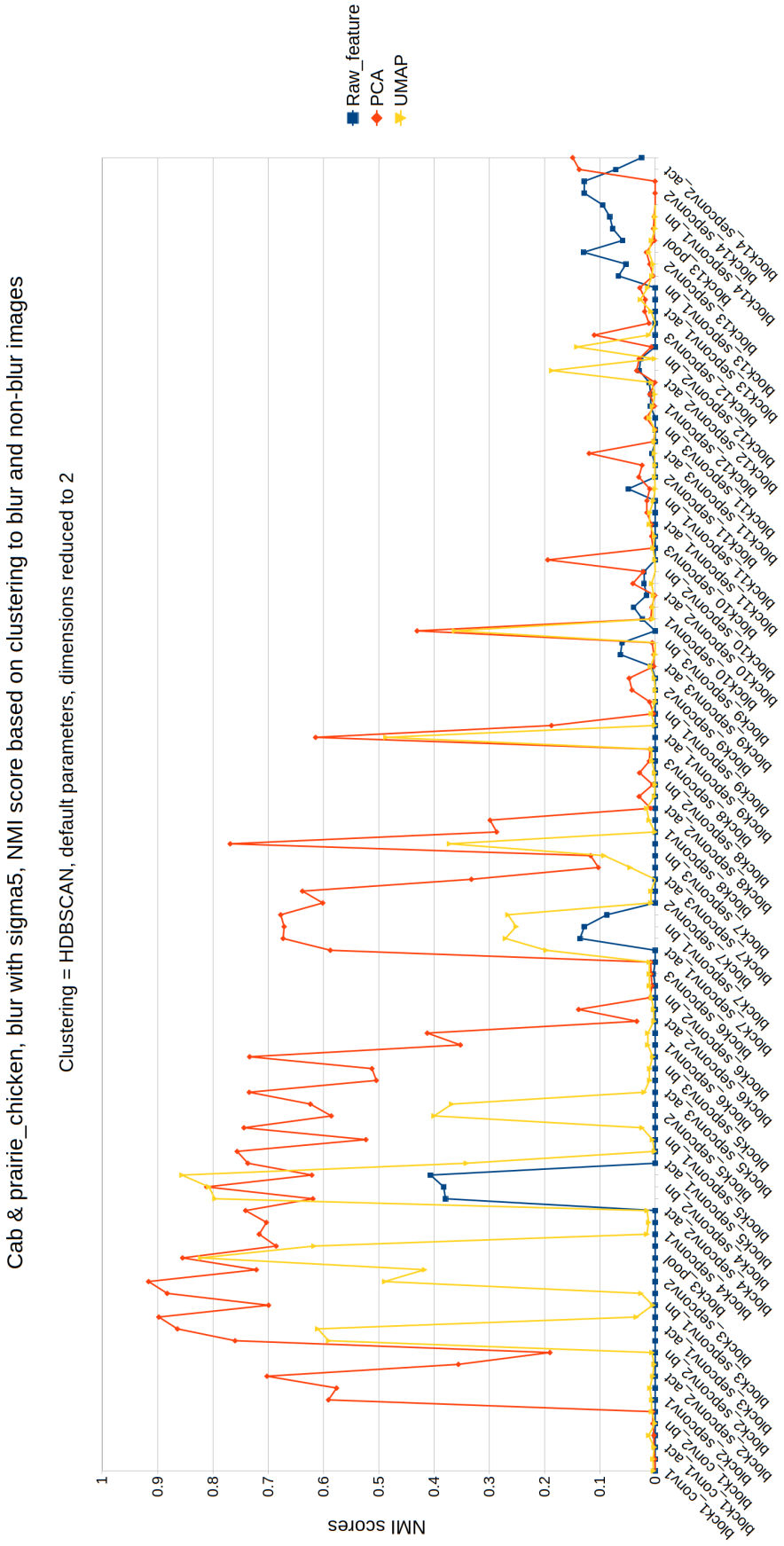

Figure 20. Cab & prairie_chicken with sigma5 blur dataset features clustered to blur and non-blur groups with original and PCA and UMAP reduced dimensions. Clustering with HDBSCAN.

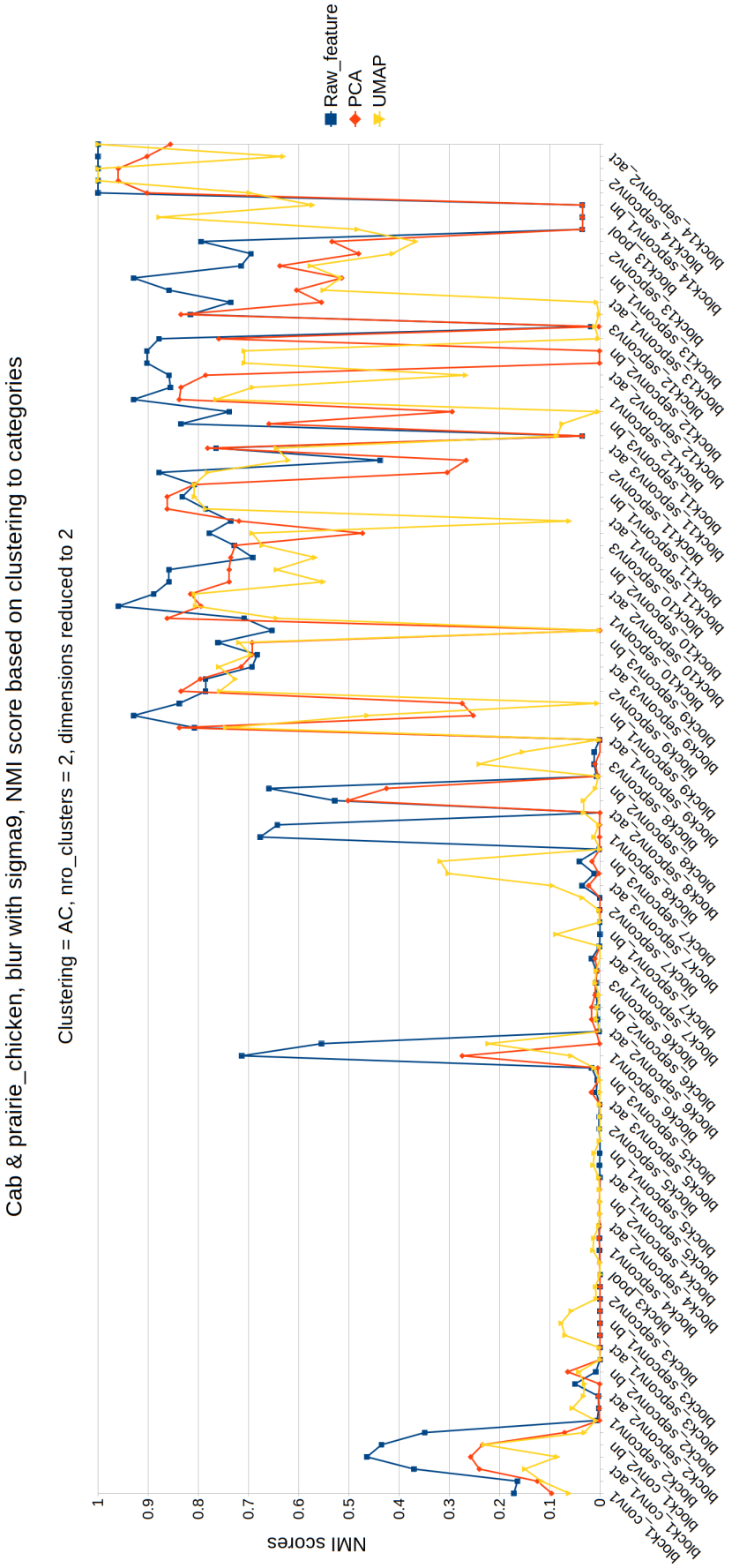

Figure 21. Cab & prairie_chicken with sigma9 blur dataset features clustered to categories with original and PCA and UMAP reduced dimensions. Clustering with AC.

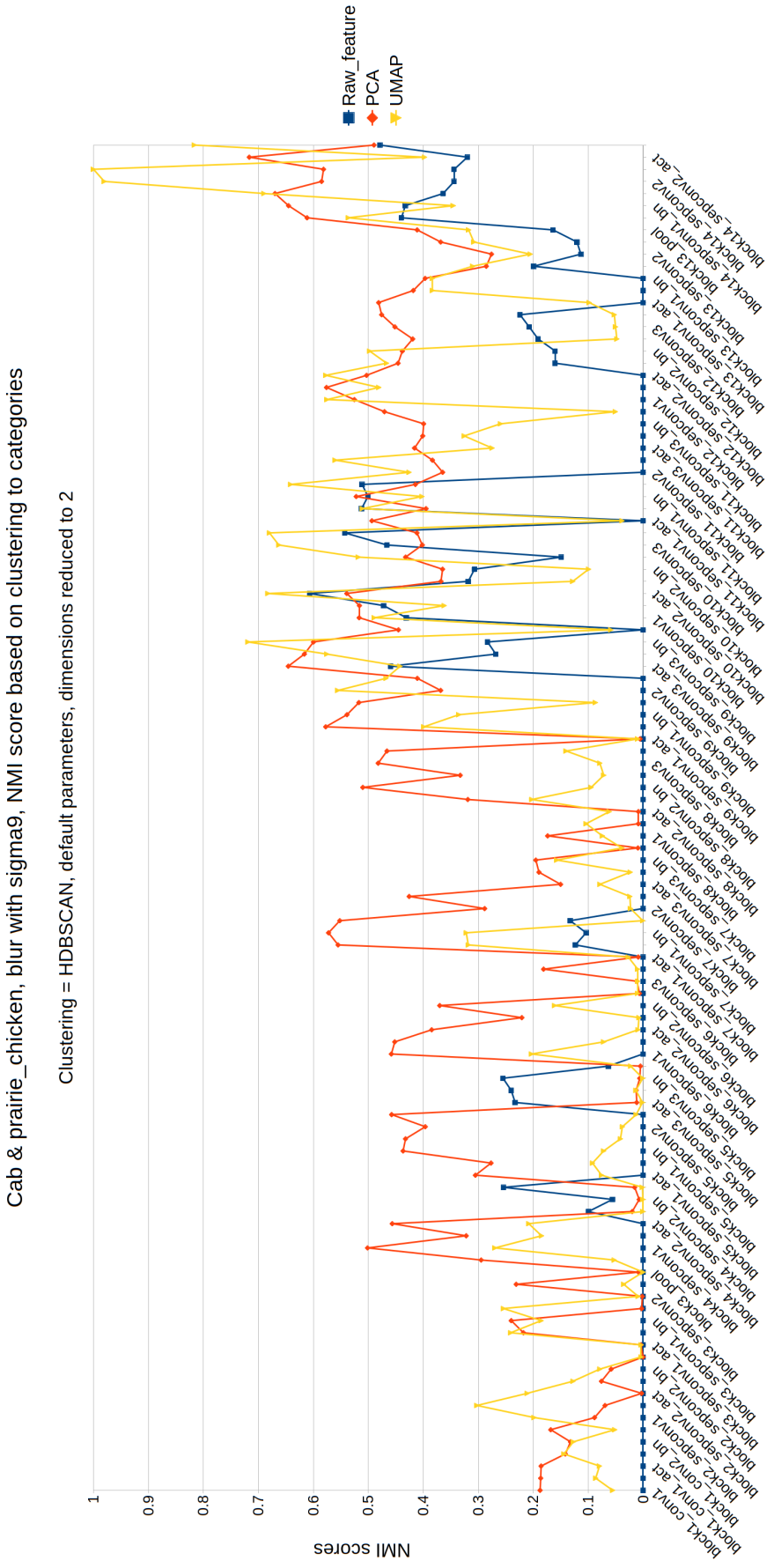

Figure 22. Cab & prairie_chicken with sigma9 blur dataset features clustered to categories with original and PCA and UMAP reduced dimensions. Clustering with HDBSCAN.

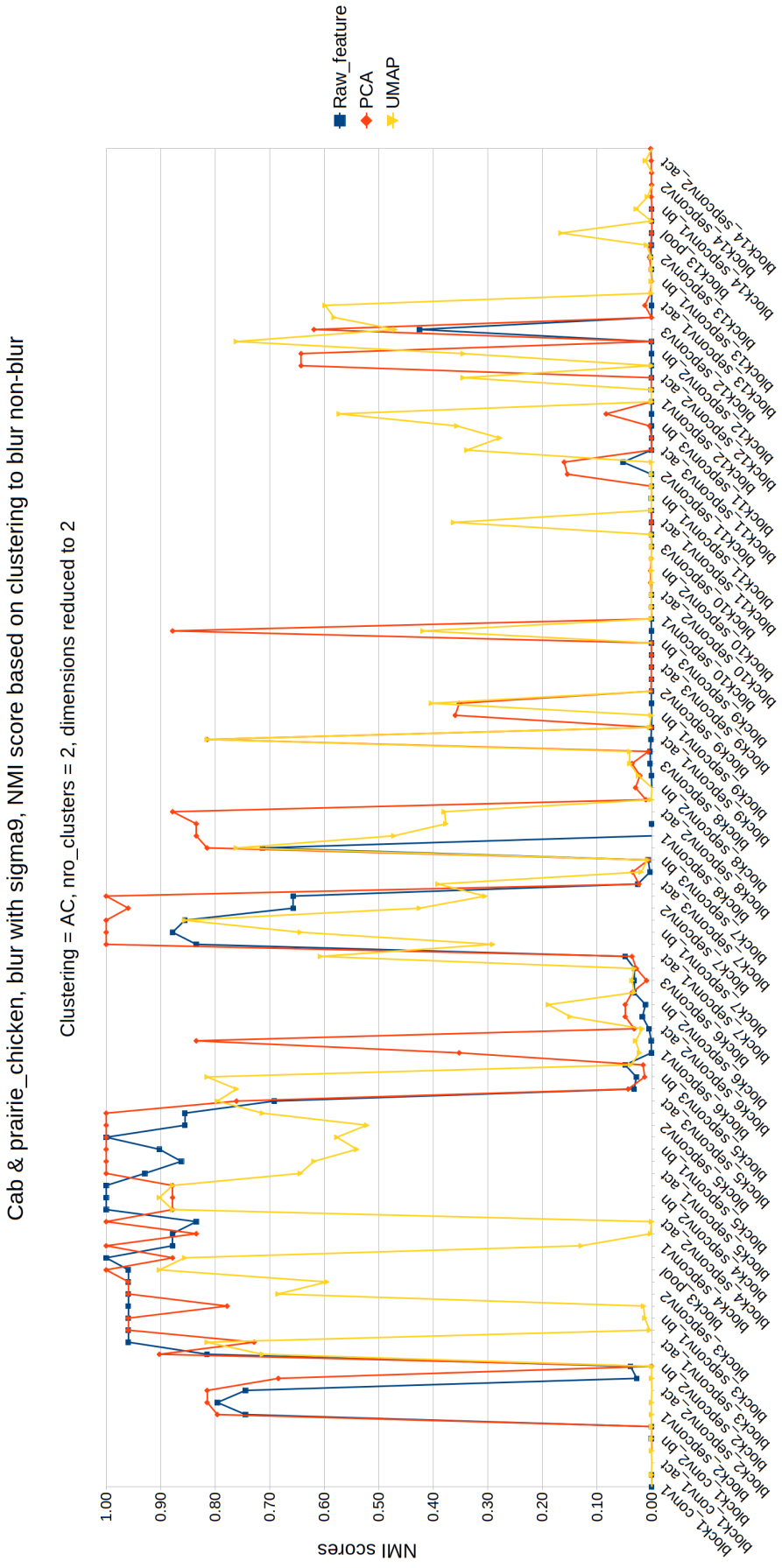

Figure 23. Cab & prairie_chicken with sigma9 blur dataset features clustered to blur and non-blur groups with original and PCA and UMAP reduced dimensions. Clustering with AC.

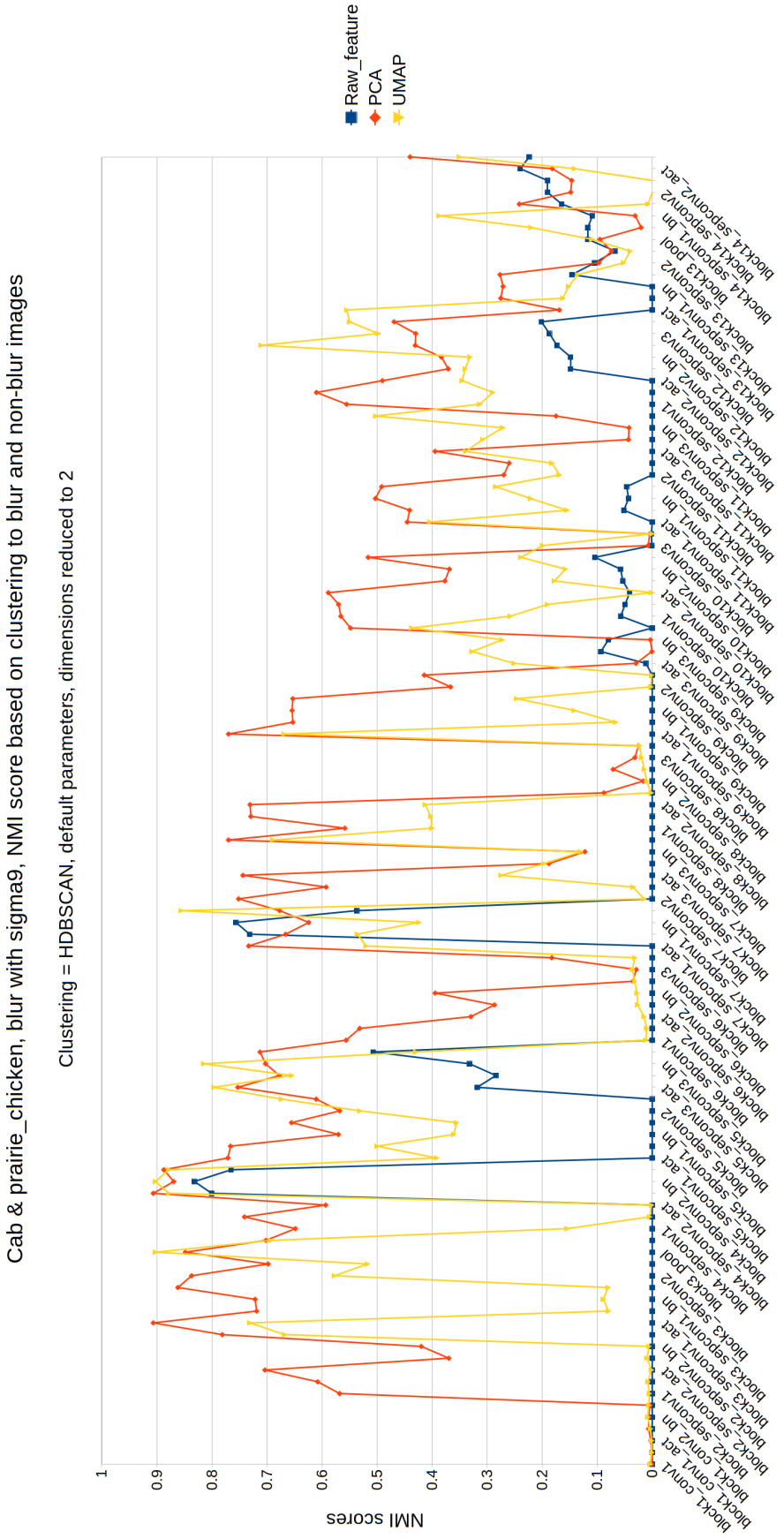

Figure 24. Cab & prairie_chicken with sigma9 blur dataset features clustered to blur and non-blur groups with original and PCA and UMAP reduced dimensions. Clustering with HDBSCAN.

Figure 1. Clustering of Cab & prairie chicken dataset with images blurred at sigma 3 on layer block14_sepconv1_bn and block14_sepconv1_act features. Correct labels set to categories of cab and prairie chicken.

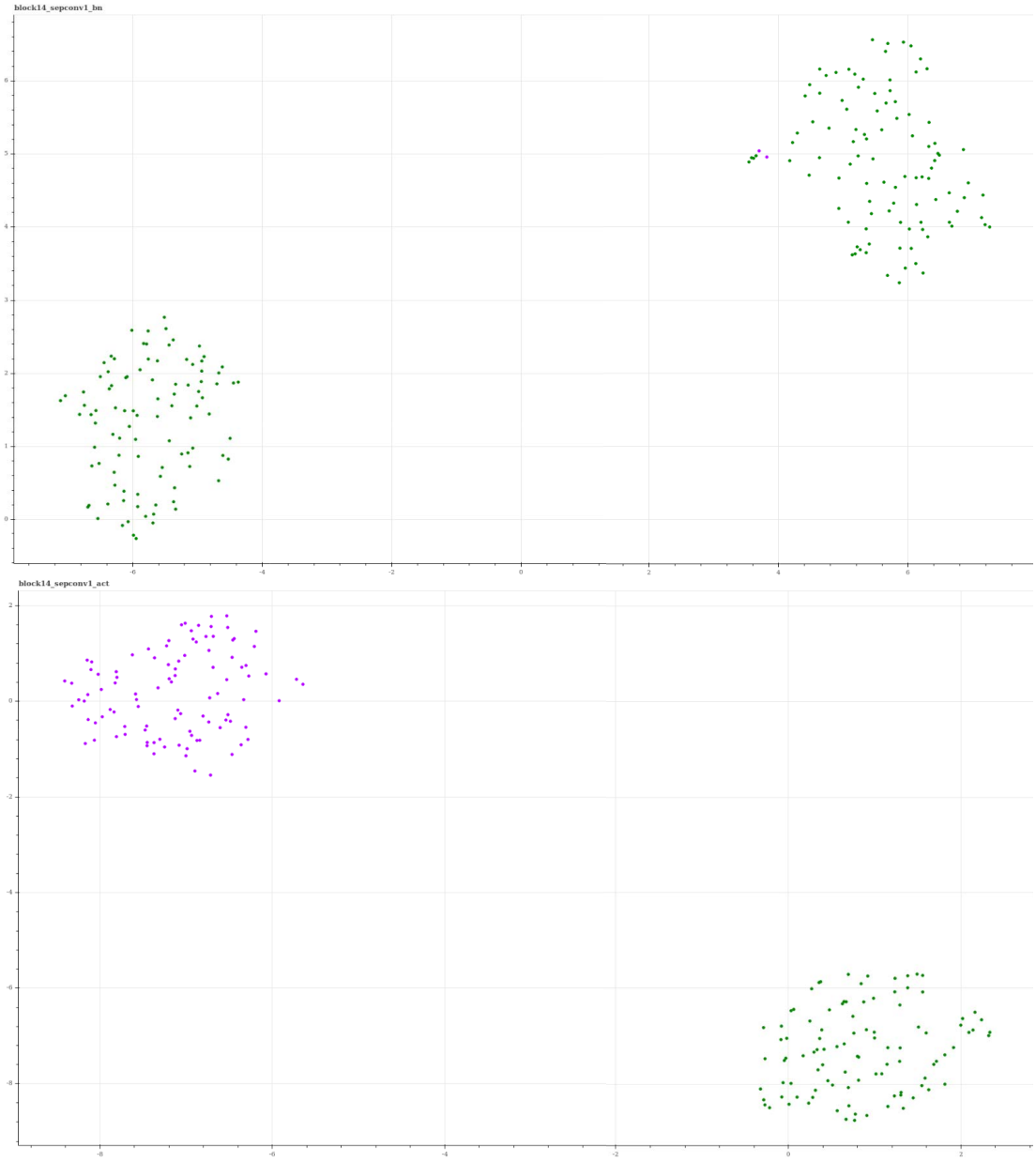

Figure 2. Clustering of Cab & prairie chicken dataset with images blurred at sigma 5 on layer block14_sepconv1_bn and block14_sepconv1_act features. Correct labels set to categories of cab and prairie chicken.

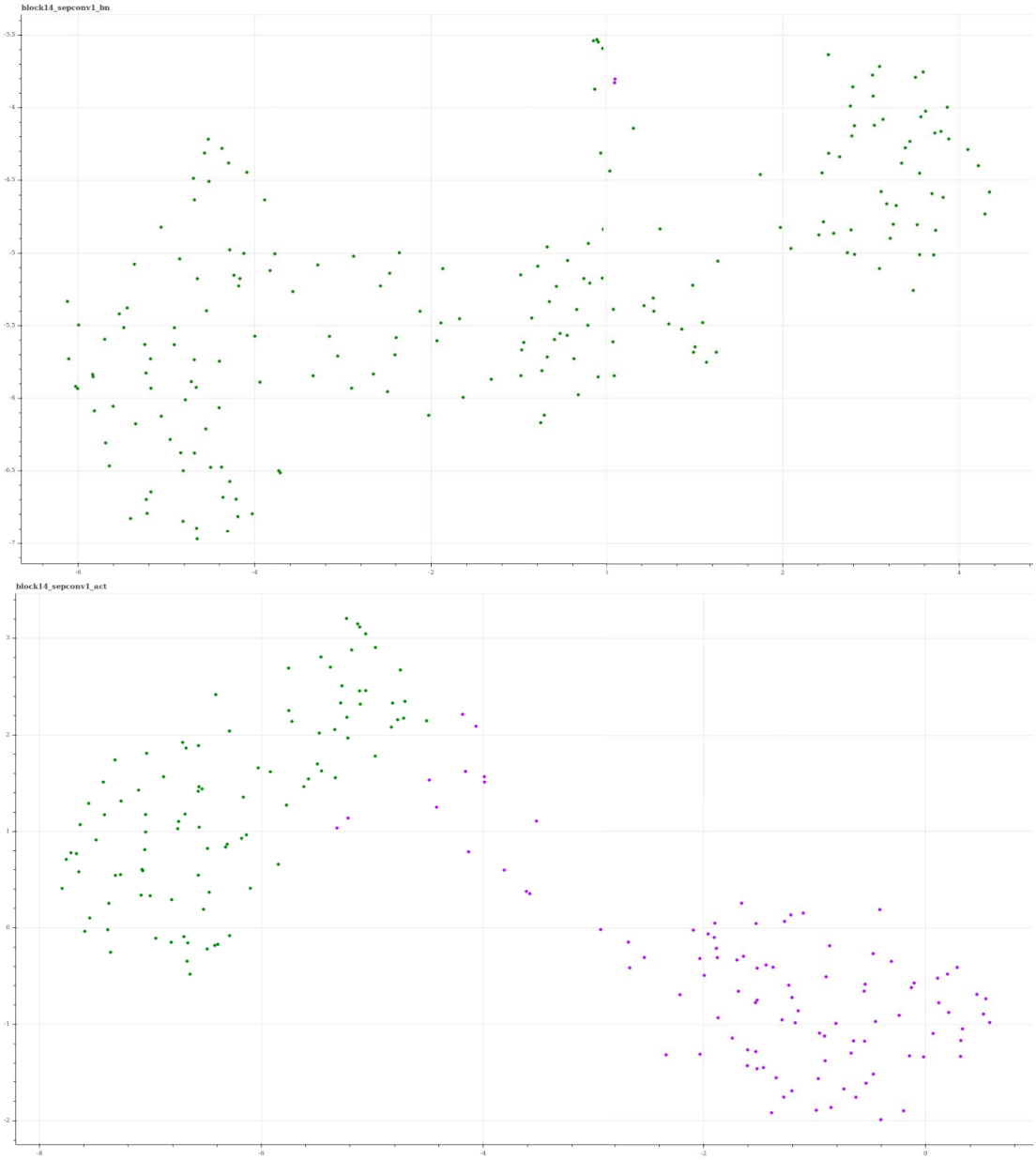

Figure 3. Clustering of *cab&chicken* dataset with images blurred at sigma9 on layer block14_sepconv1_bn and block14_sepconv1_act features. Correct labels set to categories of cab and prairie chicken.