Documentation for Project 2
Practical Machine Learning

Sîrbu Oana Adriana

Artificial Intelligence / 407

1. Short Description of My Dataset

The second project in the Practical Machine Learning course implies experimenting with various unsupervised learning methods. To achieve this, I applied a Gaussian Mixture Model (later referred to as GMM) and a Density-Based Spatial Clustering of Applications with Noise (DBSCAN) to cluster feature vectors extracted from images. The selected images are sourced from The Kaggle platform and can be accessed at this link: `https://www.kaggle.com/datasets/ifeanyinneji/donkeys-horses-zebra-images-dataset`. The image dataset is organised in 3 folders, each representing a distinct type of animals: horses, donkeys and zebras. For the purpose of this task, all images were moved into a single 'animals' folder.

For a later comparison with the number of instances in each cluster obtained, it is important to mention the number of different animals existing in our dataset:

- Horses: 384 instances
- Donkeys: 399 instances
- Zebras: 240 instances

2. Chosen Feature Extraction Methods

2.1 Using Inception Pre-trained Model

The Inception model (that is developed by Google) is widely used in the computer vision field, achieving great results in image classification tasks. This performance is due to its ability to learn distinct image features at multiple scales. It is a deep neural network that has a special architecture, containing, besides other intermediate layers, the so called Inception modules. This modules consist of parallel convolutional layers that have distinct filter sizes and pooling layers, making them really effective.

For this task, the InceptionV3 model was imported from the TensorFlow library, as follows:

```
from tensorflow.keras.applications import InceptionV3
```

To preprocess the desired images to meet InceptionV3's input requirements, the built-in **preprocess_input** method was used:

```
from tensorflow.keras.applications.inception_v3 import preprocess_input
```

The model was created using the following piece of code:

```
base_model = InceptionV3(weights='imagenet', include_top=False)
```

where weights='imagenet' implies that the pre-trained weights from the ImageNet dataset will be used and include_top=False eliminates the last layers of the deep network (those responsible for the final classification into different categories). By making this setting, we only use the model to extract relevant features.

2.2 Using Histogram of Oriented Gradients (HOG)

The histogram of oriented gradients is widely used in computer vision tasks, especially for object recognition in images. It divides each image in smaller cells, then it computes gradients within each divided cell, ultimately generating the histograms of these gradients. The histograms play an important role in capturing the shapes of local objects in images, making them a special feature for tasks like clustering.

To be able to compute such HOGs, one must import the necessary modules:

```
from skimage import io, color, feature
```

Before extracting gradient features, the image should be read from file (using **io** module) and converted to grayscale (using **color** module). Then, one must test different HOG parameters and choose the proper values for their dataset. In this case, we used:

```
hog_features = feature.hog(gray_image, orientations=8,
                           pixels_per_cell=(8,8),
                           cells_per_block=(1,1),
                           block_norm='L2-Hys', visualize=False)
```

where 'orientation=8' means that the gradients will be categorized in 8 directions, 'pixels_per_cell=(8,8)' denotes the size of each cell (in pixels), 'cells_per_block=(1,1)' means that a block will consist of a single cell and 'block_norm='L2-Hys' sets the normalization type that will be applied to the histograms in each block. These parameters were customised to achieve the best performance on the animals dataset.

The features extracted from both methods (using InceptionV3 and HOGs) were further used for training 2 clustering algorithms: GMM and DBSCAN.

Before using them for training purposes, the features were reduced to a lower dimensional representation using Principal Component Analysis (PCA). Principal components are linear combinations of original features from the given dataset, while they represent the highest variance within the data.

The PCA class can be imported from the **scikit-learn** library using:

```
from sklearn.decomposition import PCA
```

and it can be applied on our features using a **fit_transform** method.

3. Applied Models

3.1 GMM

A gaussian mixture model (GMM) is a statistical model that clusters data by identifying underlying patterns in the form of multiple Gaussian distributions, helping to organize and categorize similar data points together.

Each Gaussian component in the mixture is characterized by its mean and covariance, and the model assigns a probability to each data point belonging to a specific component. It is widely used in tasks such as image processing, this being one of the reasons for which this model was chosen for this task.

To use a GMM model, one should first import the necessary module:

```
from sklearn.mixture import GaussianMixture
```

Then, we initialize the model:

```
gmm = GaussianMixture(n_components=3, random_state=1, init_params='kmeans')
```

where we predefine the number of clusters 'n_components=3', because we are expecting the model to distinguish between those 3 types of animals: horses, donkeys and zebras, forming clusters for each kind. The 'random_state' is added for consistency, and we tested the performance of the model for both 'random' and 'kmeans' values, the latter one achieving better results.

A Grid Search was also performed on the gmm model in order to find the best value for the 'covariance_type' parameter, the options being: 'full', 'tied', 'diag' and 'spherical'. A graph representing the performance of the model (expressed by the silhouette score) on the same features obtained by the InceptionV3 is shown below:

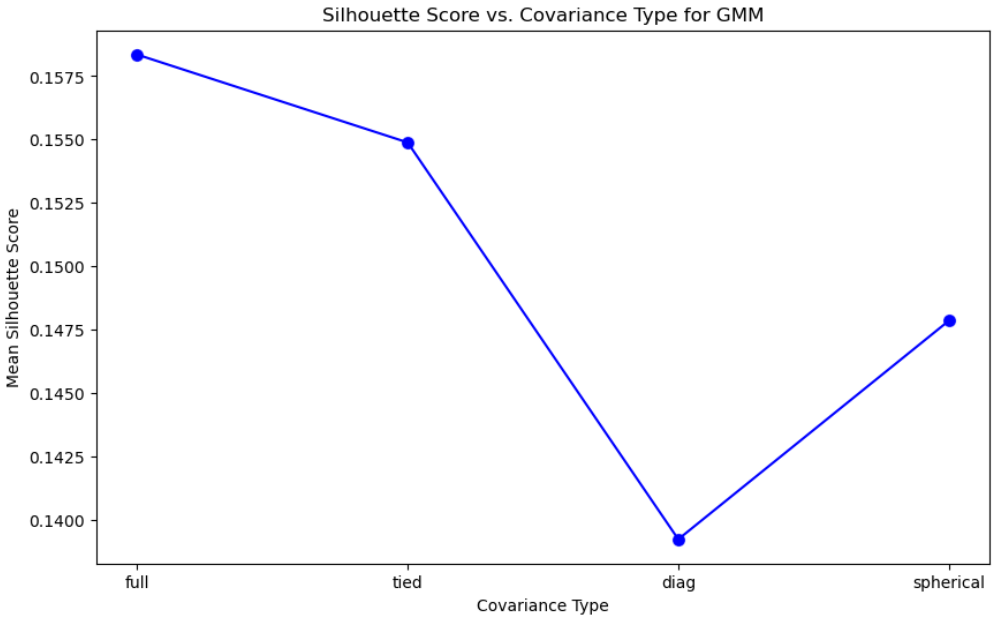

Fig 1. Silhouette score as a function of Covariance type for GMM (InceptionV3 features)

The graph suggests that the best performance is reached for the 'full' covariance type, this being also the default value of the parameter. Therefore we do not specify it in purpose in the instantiation of the gmm model.

The results were obtained using the following procedure:first, the model was trained on the features extracted using the InceptionV3 and rescaled using PCA:

```
gmm.fit(inception_features_pca)
```

The clusters were then predicted:

```
clusters_for_inception = gmm.predict(inception_features_pca)
```

We counted the number of instances assigned to each cluster:

```
cluster_counts_inc = Counter(clusters_for_inception)
print("Cluster points:", cluster_counts_inc)
```

The result being the following:

```
Cluster points: Counter({2: 501, 0: 294, 1: 228})
```

A nice illustration of these clusters, each constituent point being formed from the first and second principal components obtained after applying PCA on the InceptionV3 features can be visualized:

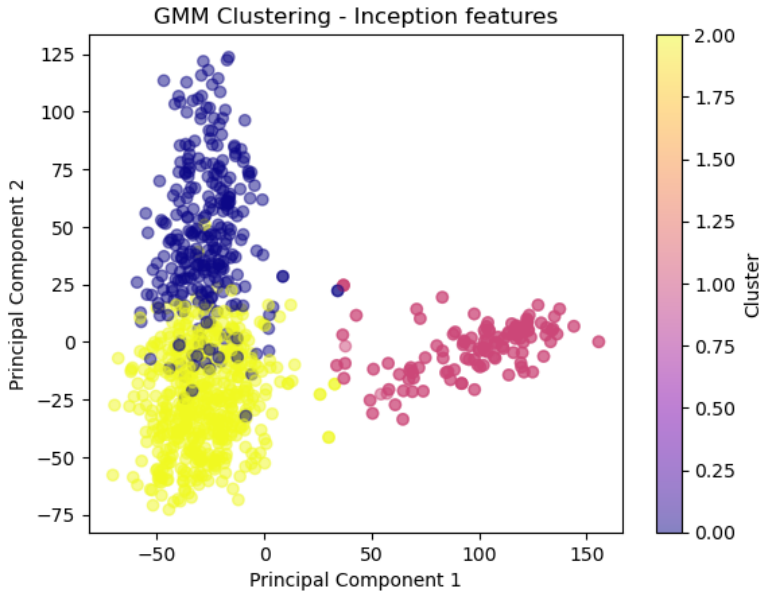

Fig 2. GMM Clustering using InceptionV3 features

The model was evaluated using Silhouette score and Calinski Harabasz score, as we can observe:

```
gmm_silhouette_inc = silhouette_score(inception_features_pca, clusters_for_inception)
print(f'Silhouette Score: {gmm_silhouette_inc}')
```

Returned:

```
Silhouette Score: 0.14685
```

And for the Calinski Harabasz score the code is similar:

```
from sklearn.metrics import calinski_harabasz_score

gmm_calinski_harabasz_inc = calinski_harabasz_score(inception_features_pca,
                                                    clusters_for_inception)
print(f'Calinski-Harabasz Index: {gmm_calinski_harabasz_inc}')
```

This time the result being:

```
Calinski-Harabasz Index: 167.36316
```

The GMM model was also applied on the HOG features (hog_features_pca). The code structure is the same as for the InceptionV3 features. The new results will be presented. First, a new GridSearch was computed, but the same best parameters as before were kept.

```
Cluster points: Counter({1: 628, 2: 299, 0: 96})
```

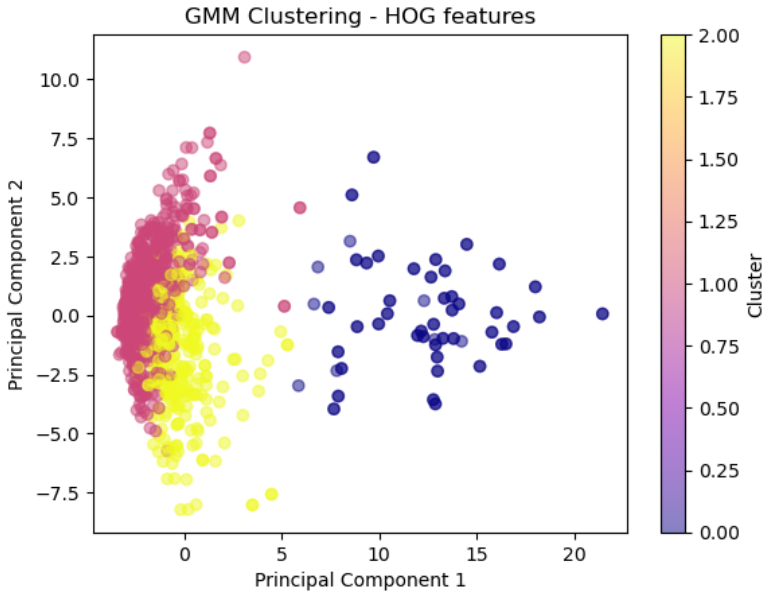

Fig 3. GMM Clustering using HOG features

```
gmm_silhouette_hog = silhouette_score(hog_features_pca, gmm_clusters_for_hog)
print(f'Silhouette score: {gmm_silhouette_hog}')
```

The result for the silhouette score being, this time:

```
Silhouette score: 0.13398
```

And using the other metric:

```
gmm_calinski_harabasz_hog = calinski_harabasz_score(hog_features_pca,
                                                    gmm_clusters_for_hog)
print(f'Calinski-Harabasz Index: {gmm_calinski_harabasz_hog}')
```

The new evaluation score is:

```
Calinski-Harabasz Index: 180.9279
```

3.2 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise), is an algorithm used for recognizing dense regions within a dataset by analyzing the distribution of data points in the feature space. As opposed to conventional approaches that need the cluster count set before training, DBSCAN automatically forms clusters

with various shapes and/or sizes. One can use it even when we expect clusters to have irregular shapes.

To use a DBSCAN model, one should first import the necessary module:

```
from sklearn.cluster import DBSCAN
```

A new model can be initialised using the following code:

```
dbscan = DBSCAN(eps=0.4, min_samples=30, metric='cosine')
```

This time, we won't set the number of clusters this model should predict as one of the initial parameters. The number of clusters actually depends on the **eps** and **min_samples** parameters. For this project, the values of these parameters were chosen after performing a Grid Search. The search was made with these inputs: 'eps': [0.2, 0.3, 0.4, 0.5, 0.6], 'min_samples': [5, 10, 15, 20, 25, 30, 35], 'metric': ['euclidean', 'cosine'].

The result of:

```
print(best_params_dbscan)
```

is:

```
{'eps': 0.4, 'metric': 'cosine', 'min_samples': 30}
```

The evaluation was also computed based on the mean silhouette scores of a DBSCAN trained on InceptionV3 features. A heatmap will allow a better understanding of the dependence of the silhouette score on different combinations of the 'eps' and 'min_samples' parameters:

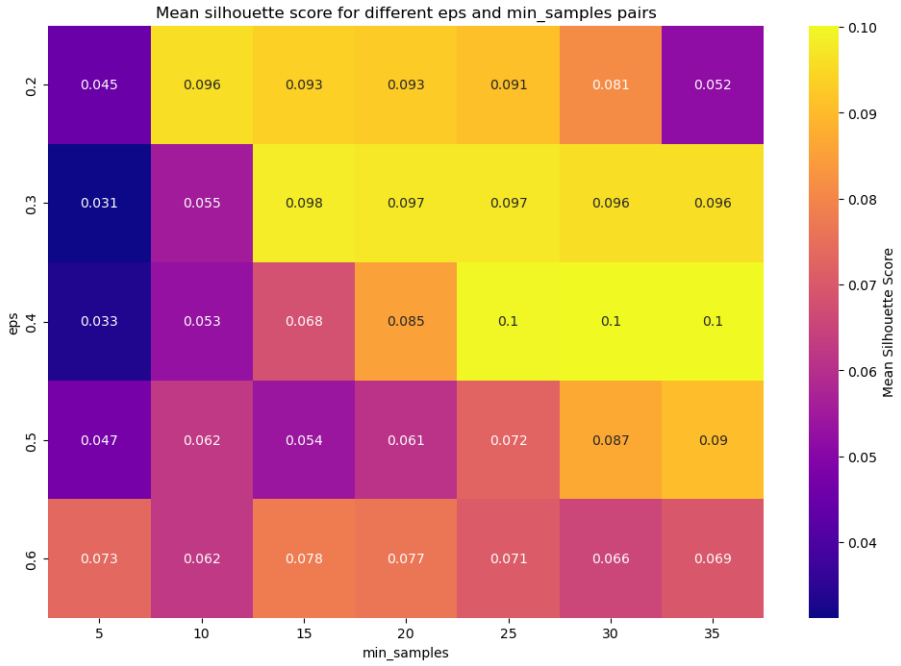

Fig 4. Mean silhouette score for different eps and min_samples pairs

A DBSCAN model can be trained on both InceptionV3 and HOG features, as we previously saw for the GMM. The code structure is similar:

```
clusters_for_dbscan_inc = dbscan.fit_predict(inception_features_pca)
```

The number of cluster points obtained after training on PCA InceptionV3 features being:

```
Cluster points: Counter({-1: 608, 0: 230, 1: 185})
```

A visualisation of these clusters can be easily made:

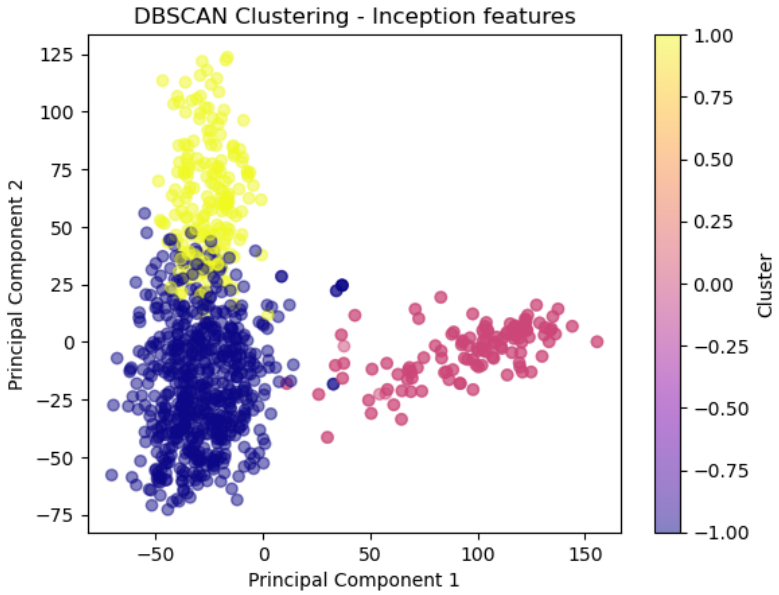

Fig 5. DBSCAN Clustering - InceptionV3 features

The Silhouette score for this approach is 0.14849, while the Calinski-Harabasz score is 165.07268.

Using the PCA HOG features, the performance of the DBSCAN (with the same parameters as before, it proved to be the best) changes:

```
Cluster points: Counter({0: 670, -1: 293, 1: 60})
```

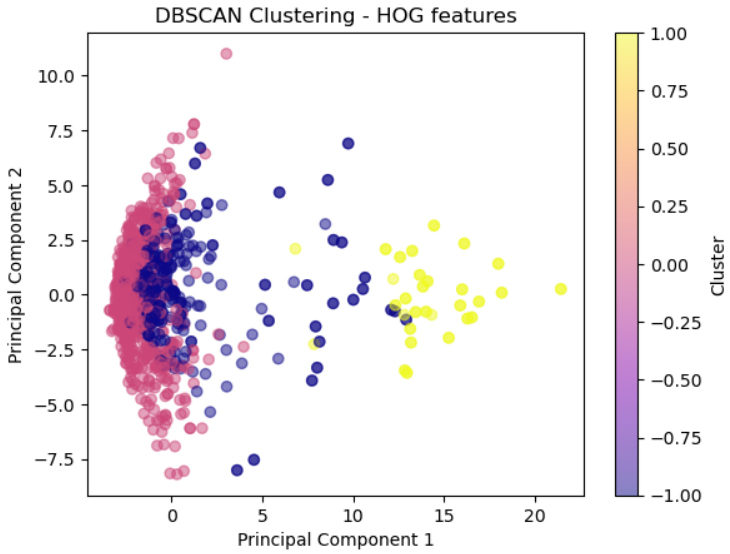

Fig 6. DBSCAN Clustering - HOG features

For this method, the evaluation metrics become: Silhouette score = 0.17883, Calinski-Harabasz score = 129.89227.

4. Comparison with Random Choice and a Supervised Baseline

4.1 Random Choice

For the comparison with random choice, the following approach was used:

We used a Dummy classifier from the scikit-learn library:

```
from sklearn.dummy import DummyClassifier
```

The dataset was divided in training and test sets using a **train_test_split()** function.

An instance of this Dummy classifier was created:

```
dummy_classifier_comp = DummyClassifier(strategy='stratified')
```

It was fitted on the training features and tested on the test ones:

```
dummy_classifier_comp.fit(features_train_dummy, labels_train_dummy)
labels_pred = dummy_classifier_comp.predict(features_test_dummy)
```

To evaluate its performance, a classification report was generated:

	precision	recall	f1-score	support
donkey	0.35	0.43	0.38	65
horse	0.40	0.39	0.40	79
zebra	0.36	0.28	0.31	61
accuracy			0.37	205
macro avg	0.37	0.37	0.37	205
weighted avg	0.37	0.37	0.37	205

The **features_train_dummy** were changed from the Inception features to the HOG ones and the same structure was followed.

The classification report this time looks like:

	precision	recall	f1-score	support
donkey	0.32	0.45	0.37	65
horse	0.40	0.39	0.39	79
zebra	0.33	0.20	0.25	61
accuracy			0.35	205
macro avg	0.35	0.35	0.34	205
weighted avg	0.35	0.35	0.34	205

Out of curiosity, another random choice method was tested:

```
num_samples = 1023
random_choice_labels = np.random.randint(0, 3, num_samples)
random_choice_silhouette_score = silhouette_score(inception_features_pca,
                                 random_choice_labels)
random_choice_ch_score = calinski_harabasz_score(inception_features_pca,
                                 random_choice_labels)
```

A number of 1023 artificial random labels (between 0 and 2) were generated, and the silhouette and calinski harabasz scores were computed for comparison purposes.

The results are:

```
Random silhouette score: -0.006058213301002979
Random calinski_harabasz_score: 0.9424546036571576
GMM silhouette score: 0.14685191214084625
GMM calinski_harabasz_score: 167.3631609105279
DBscan Silhouette Score: 0.14849881827831268
DBScan calinski_harabasz_score: 165.0726887780126
```

Both GMM and DBSCAN demonstrated a better performance than a random choice approach.

4.2 Supervised Baseline

Due to the fact that the original data was labeled, we could also make a comparison with a supervised model. For this project, a Random Forest Classifier was trained on the same features (from InceptionV3 and HOG), in order to see its performance and test its capability of making classifications.

In order to correctly test the results of the supervised algorithm, the original PCA features from InceptionV3 model were splitted in features_train and features_test.

The performance of the Random Forest Classifier can be analysed with the following metrics:

```
Classification Report:
              precision    recall  f1-score   support

      donkey       0.83      0.91      0.87        65
       horse       0.92      0.84      0.87        79
       zebra       0.98      1.00      0.99        61

    accuracy                           0.91       205
   macro avg       0.91      0.91      0.91       205
weighted avg       0.91      0.91      0.91       205
```

This model has proven to be highly effective in classifying the extracted features.

On the other hand, the GMM model applied on the same features_train and that was asked to predict on the features_test the labels_test, reached the following score:

```
Cluster points: Counter({1: 102, 0: 58, 2: 45})
Silhouette score: 0.14072783291339874
```

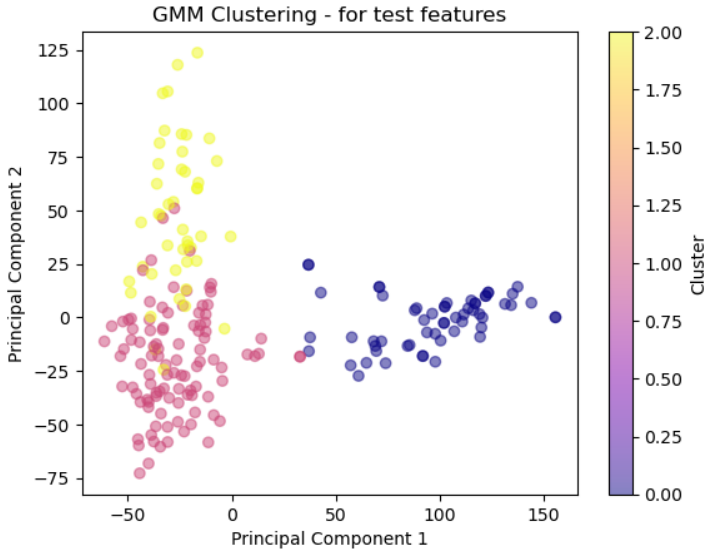

Fig 7. GMM clustering - for test features

We can compare these 2 approaches by analyzing the number of instances corresponding to each class/-cluster. It can be observed that, although the unsupervised algorithm doesn't perform as well as the supervised one (as expected), the distribution of instances across the classes/clusters doesn't vary significantly.

The same tests were also run on the HOG features. This time, the Random Forest Classifier's performance is:

Classification Report:

	precision	recall	f1−score	support
donkey	0.52	0.66	0.58	65
horse	0.67	0.61	0.64	79
zebra	1.00	0.82	0.90	61
accuracy			0.69	205
macro avg	0.73	0.70	0.71	205
weighted avg	0.72	0.69	0.70	205

One can observe that all the metrics have lower values than for the previous set of features. We can conclude that the Random Forest Classifier model performs worse on the HOG features. A GMM model was also trained on the same set on training features, such that the comparison makes sense.

The number of instances for each cluster obtained this time is:

```
Cluster points: Counter({0: 130, 1: 58, 2: 17})
```

while the silhouette score is:

```
Silhouette score: 0.149747
```

The clusters can be observed in the following figure:

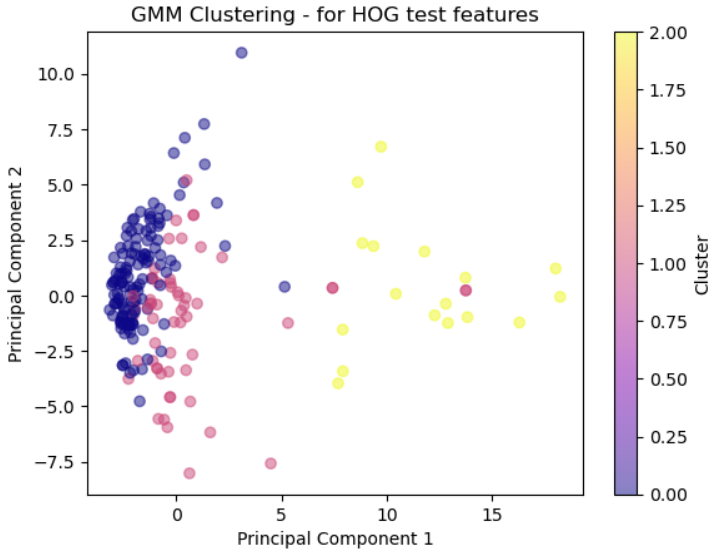

Fig 8. GMM clustering - for HOG test features

From this results, we can affirm that both the supervised (through Random Forest Classifier) and the unsupervised (GMM) perform better on the InceptionV3 features than on the HOG ones.

5. Additional model tested

To confirm the fact that the InceptionV3 captured better features, another model was tested, namely K-means. K-means is a clustering algorithm that groups similar data points together. In simple terms, it repeatedly adjusts cluster centers to minimize the distance between data points and their assigned clusters.

In python, one can use this model by importing the necessary module:

```
from sklearn.cluster import KMeans
```

Because we already knew that our data should be divided in 3 categories, we set the number of clusters to be equal to three. An instance of the model will look like:

```
kmeans = KMeans(n_clusters=3, random_state=1)
```

In a similar manner to what we saw at the GMM and DBSCAN models, we can fit and predict the Inception extracted features:

```
clusters_for_kmeans_inc = kmeans.fit_predict(inception_features_pca)
```

We obtain the following results:

```
Cluster points: Counter({1: 544, 0: 251, 2: 228})
Silhouette score: 0.154423
Calinski-Harabasz score: 172.27899
```

The clusters have the following shapes (similar to those obtained by GMM):

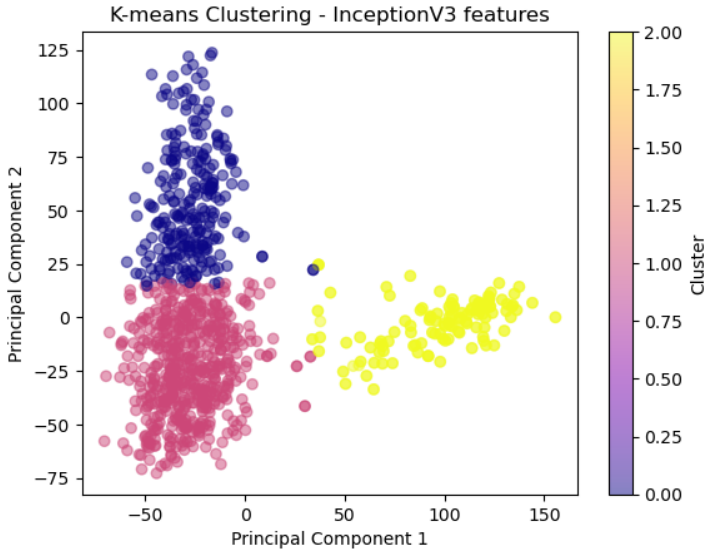

Fig 9. K-means Clustering - InceptionV3 features

When applied on the HOG features, the model performance was:

```
Cluster points: Counter({1: 482, 0: 443, 2: 98})
Silhouette score: 0.09234
Calinski–Harabasz score: 199.48867
```

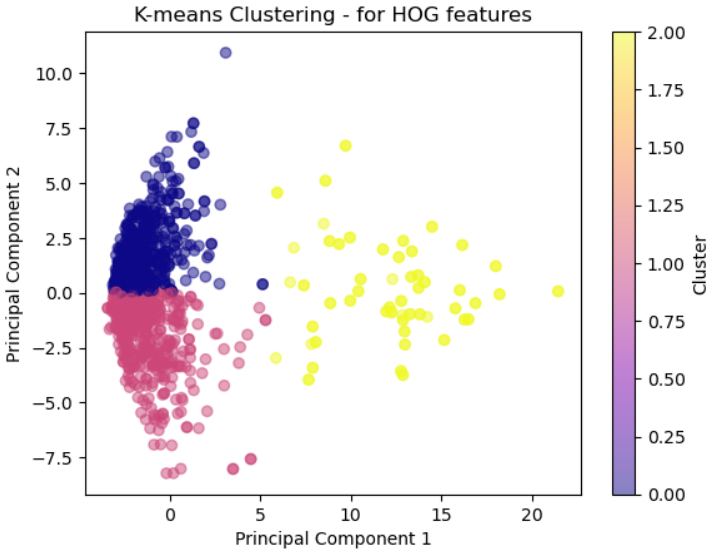

Fig 10. K-means Clustering - for HOG features

6. Conclusions

In summary, all unsupervised models (GMM, DBSCAN and K-means) successfully represented the input images as three different clusters. All models indentified 2 clusters with similar features, likely corresponding to horses

and donkeys, while the third class is distinguished by the very different principal components resulted after the PCA dimensionality reduction. When compared to the random choice method, the unsupervised algorithms performed better. On the other hand, the supervised model, which used labeled data for each feature, achieved the highest performance, with an accuracy of over 90% for Inception features and nearly 70% for HOG features. These results illustrate the capacity of the deep neural network in capturing better features. The same thing was also observed for the unsupervised models, which shaped better the clusters when trained on the Inception features.