Final Work in Machine Learning

Ilan Meyer Souffir : 342615648

Shai Moshe : 318160108

Link project github : https://github.com/ilan2505/FinalWork_ML.git

Our code is the file : project_ML.ipynb

Requirements:

pandas
seaborn
tqdm
zipfile
pathlib
numpy
matplotlib
tensorflow
sklearn
keras

1. Dataset

Our Dataset is from:
https://www.kaggle.com/datasets/ponrajsubramaniian/sportclassificationdataset
and we decided to modify it to have now a set of 3059 sports images, divided into 7 folders: baseball, basketball, boxing, football, formula1, swimming and tennis.
The dataset is located on google drive as a zip file:
https://drive.google.com/file/d/1dmX5laQwy4JYHSPHO2H7MX6JYzKRH7iO/view?usp=sharing
The number of baseball, basketball, boxing, football, formula1, swimming and tennis images is for each category 437 images.
The average images size is 476 x 322.

2. Process

In this part, we want to split the dataset into training and testing: 80% for training and 20% for testing but you can choose what you want. We also want an image of 128x128 because it's better for our accuracy results. Then we need to modify and normalize the images to have better images to process and at the end we flatten the images.

2.1 Split the dataset into training and testing.

Images are adjusted to a size of 128x128 pixels, with the color scheme being switched to RGB. This resizing uses the Lanczos5 interpolation technique, which is advised for reducing the size of images. So we now use 2448 files for training and 611 for validation.

2.2 Adjustment and Normalization of the images.

Each image is 128x128 pixels with 3 channels (rgb). The labels are one-hot encoded. In this part, we will use the keras preprocessing layers to get a better image with the following techniques:

- RandomZoom : Randomly zooms the image
- RandomContrast : Randomly changes the contrast of the image
- Normalization : Normalizes the image

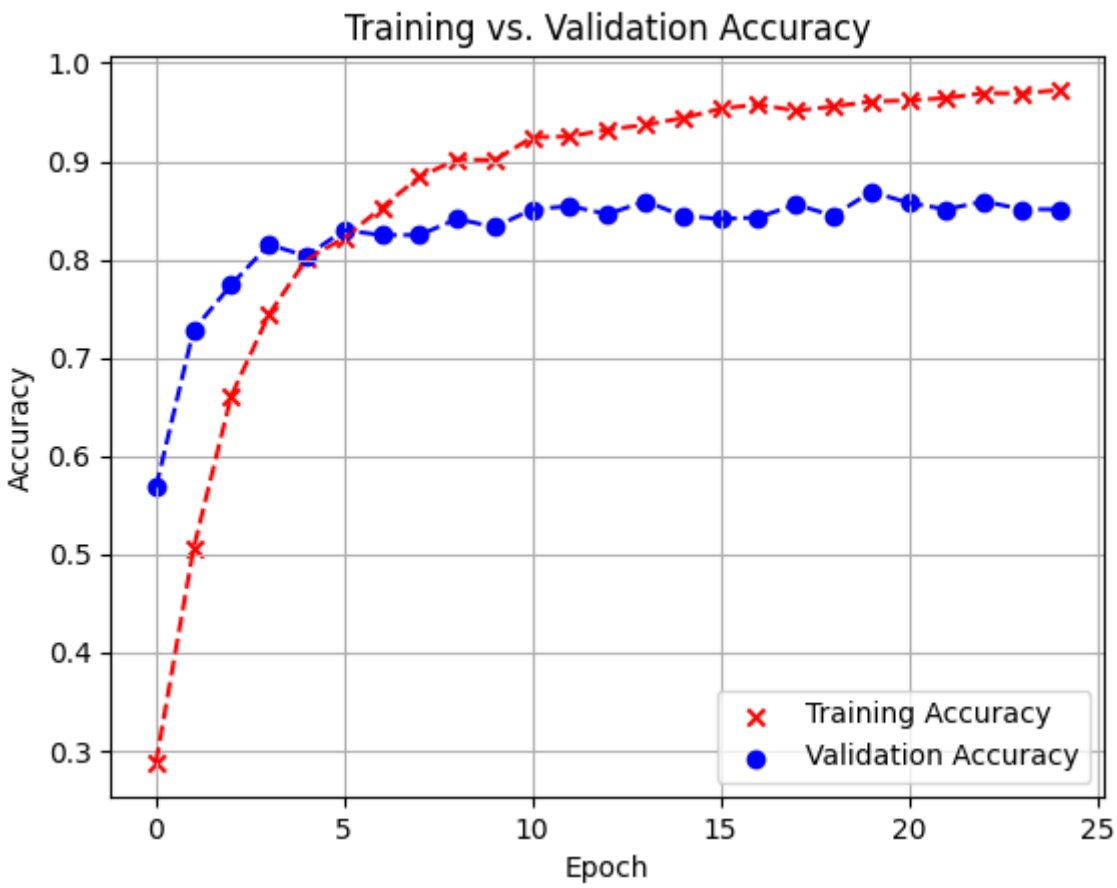
2.3. Flatten the images using VGG16

We will use the pre-trained VGG16 model on ImageNet to flatten the images. The VGG16 is a conventional neural network (CNN) architecture for image classification, it prepares an image dataset for further training by extracting features via the model and normalizing pixel values, all while retaining the original structure and weights of the VGG16 model for feature extraction.It is recommended for vectorizing images.

3. Build the model

3.1 Linear SVM

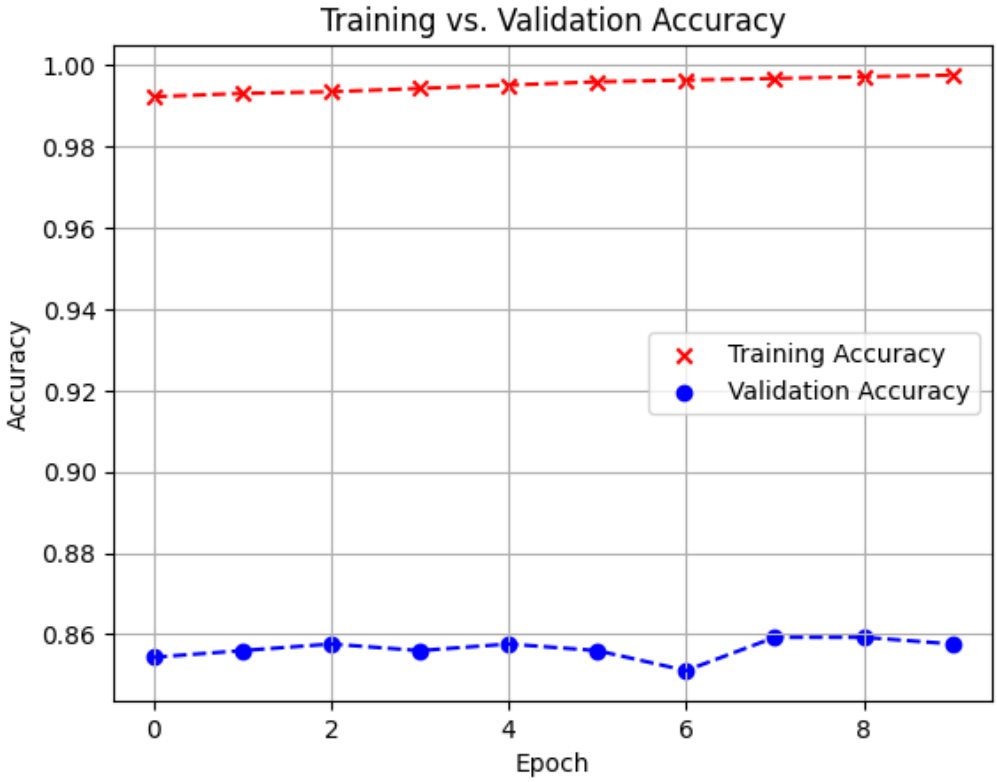
The Linear SVM model operates as a linear classifier, employing the hinge loss function to enhance the margin separating the classes. This model is constructed using several dense layers, interspersed with dropout layers. The incorporation of multiple dense layers aids in delineating a more intricate decision boundary. Meanwhile, the dropout layers are instrumental in mitigating overfitting. Within this framework, the squared hinge loss function, defined as:
$\mathrm{L}(\mathrm{y}, \mathrm{f}(\mathrm{x})) = \max(0,1 - \mathrm{y} * \mathrm{f}(\mathrm{x}))^2$, is utilized.
This particular function imposes a greater penalty on incorrectly classified points. Our experimentation indicates that this loss function outperforms the traditional hinge loss function in effectiveness.



1. **Accuracy Increase**: The model's training accuracy starts at 28.76% and consistently increases to reach 97.26% by the last epoch. Similarly, the validation accuracy improves from 56.79% to 85.11%. These are strong indicators of the model learning effectively from the training data.
2. **Loss Reduction**: Both the training and validation loss decrease over time. The training loss goes from 1.2093 to 0.8820, and our validation loss decreases from 1.1158 to 0.9475. A decreasing loss is a good sign that the model is converging and becoming more confident in its predictions.
3. **Validation Performance**: The validation accuracy and loss are good indicators of how well the model generalizes to unseen data. In this case, the validation accuracy ends at a high note (85.11%), suggesting that the model has learned generalizable features rather than just memorizing the training data.
4. **Potential Overfitting**: While the training accuracy reaches a very high level (97.26%), validation accuracy caps at 85.11%. This gap suggests a bit of overfitting, where the model performs significantly better on the training data than on unseen data.

3.2. Clustered Linear SVM

Primarily aimed at model compression to minimize its size, the Clustered Linear SVM technique also significantly enhances model performance. By clustering the model's weights, it reduces the parameter count, achieving a leaner model that not only demands less storage but also operates more efficiently. This approach not only streamlines the model for better deployment but also potentially boosts its accuracy and performance, making it a valuable strategy for optimizing machine learning models where efficiency and effectiveness are paramount.



1. **High Starting Accuracy**: The model begins with a remarkably high training accuracy (99.75%) and validation accuracy (85.76%). The high starting point suggests that the model, possibly due to pre-processing with clustering or initialization, begins in a state that is already highly effective at making predictions.
2. **Minimal Accuracy Improvement**: Across epochs, the training accuracy slightly increases from 99.22% to 99.75%, and the validation accuracy shows minor fluctuations around the 85% to 86% range. These small changes imply that the model was already near its maximum capability from the beginning of training.
3. **Decreasing Loss**: The training loss decreases from 0.0594 to 0.0157, which is good, but given the starting loss was already low, the relative improvement is marginal. The validation loss also decreases slightly before stabilizing, indicating the model's predictions are becoming slightly more confident over time without significantly changing in accuracy.
4. **Validation Performance vs. Training Performance**: The gap between training and validation accuracy is more pronounced than in the previous SVM model, with training accuracy near 100% and validation accuracy in the mid-80s percent range. This discrepancy suggests a higher degree of overfitting, where the model is almost perfectly fitting the training data but not improving as much on the validation set.
5. **Potential Overfitting Concerns**: The very high training accuracy combined with the much lower validation accuracy suggests that the model might be overfitting to the training data.

3.3. AdaBoost with Decision Tree Classifier

In this approach, we integrate AdaBoost with a decision tree classifier as the foundational model. Utilizing RandomizedSearchCV, we aim to pinpoint the optimal hyperparameters, enhancing the model's efficiency. AdaBoost, known for its applicability to both classification and regression tasks, is selected for its ability to augment model performance significantly. This boosting algorithm improves upon the base decision tree model by iteratively correcting errors, thus offering a more accurate and robust solution.

Accuracy: 0.6380718954248366
Best score: 0.5261437908496732
Best hyperparameters: {'n_estimators': 200, 'learning_rate': 0.3}

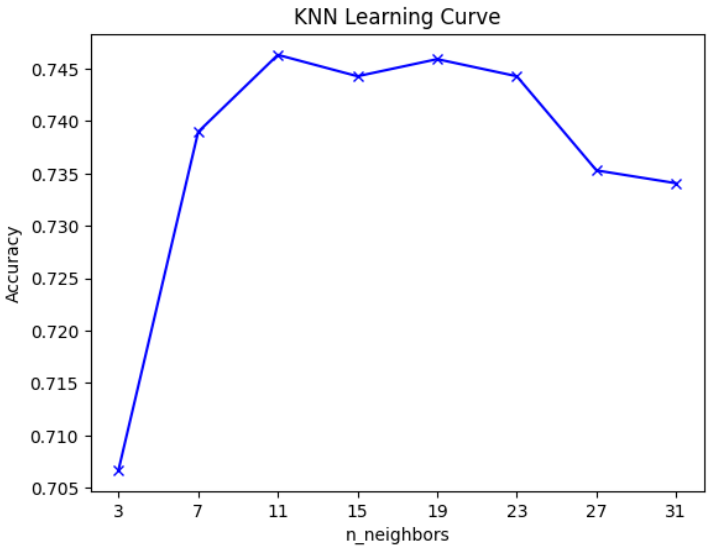
3.4. AdaBoost with Neural Network

In this configuration, we combine AdaBoost with a neural network as the base model, employing RandomizedSearchCV to efficiently identify optimal hyperparameters. AdaBoost enhances model accuracy by iteratively focusing on challenging cases, while the neural network's flexibility allows for complex pattern recognition. RandomizedSearchCV, by sampling a subset of hyperparameter combinations, offers a balance between exploration and computational efficiency, making it ideal for fine-tuning the sophisticated architecture of neural networks in this boosted ensemble method.

Accuracy: 0.8527004909983633
Best score: 0.8051470588235294
Best hyperparameters: {'n_estimators': 10, 'learning_rate': 0.01, 'base_estimator__epochs': 5, 'base_estimator__batch_size': 64}

3.5. K-Nearest Neighbors

K-Nearest Neighbors stands as a straightforward algorithm known for its commendable performance capabilities. In our implementation, we opt for Cosine Similarity as the distance metric, given its effectiveness with high-dimensional data. To ensure the model operates at its peak, we employ RandomizedSearchCV, a method designed to meticulously search and identify the optimal hyperparameters for the algorithm.



Accuracy: 0.8051470588235294
Best score: 0.7463235294117648
Best hyperparameters: {'n_neighbors': 11}

The KNN model is performing reasonably well, with the highest cross-validation scores around the mid-70s percentage mark. However, there's always room for improvement through more detailed parameter tuning, feature engineering, or incorporating domain-specific knowledge.

3.6. Logistic Regression

Logistic regression is a supervised classification algorithm that models the probability an observation belongs to a specific category using a logistic function. Our implementation employs logistic regression to classify observations into multiple categories, optimizing hyperparameters through random search and evaluating performance using metrics like accuracy and the F1 score on a test set.

Accuracy : 0.9995915032679739
Best Score: 0.8227124183006537
Best hyperparameters: {'penalty': 'l2', 'C': 1}
Accuracy on the test: 0.851063829787234
Confusion Matrix : tf.Tensor(
[[55 4 1 5 1 1 5]
 [1 77 5 2 7 0 0]
 [0 3 86 1 3 0 3]
 [2 3 1 75 4 1 4]
 [1 0 1 2 71 1 0]
 [0 2 0 1 1 87 0]
 [5 8 5 5 2 0 69]], shape=(7, 7), dtype=int32)

1. **Model Performance by Hyperparameter Configuration:** The results show performance scores (accuracy) for different combinations of C and penalty type (l1, l2) across 3 folds of cross-validation. For instance, with C=1 and penalty=l1, the scores are roughly 75%, 80%, and 79%.
2. **Overall Evaluation:** The best overall validation score achieved is about 82.27% with C=1 and penalty=l2, indicating this combination of hyperparameters performs the best out of the tested configurations on the validation sets.
3. The **accuracy** on the test set is approximately 85.11%, and the detailed confusion matrix provides insight into the classification performance across the 7 classes. The matrix reveals how each class was predicted, showing the true positives along the diagonal and the misclassifications off-diagonal.
4. **Insights:** The relatively high accuracy and the confusion matrix suggest the model performs well, though there are misclassifications. Fine-tuning, feature engineering, or exploring other models could potentially improve performance.

4. Evaluation and Questions

4.1. Evaluation

For this part we are interested in two parameters:

- **accuracy**: is a metric used to evaluate the overall performance of a classification model. The formula is :

$$\text{Accuracy} = \frac{\text{NumberofCorrectPredictions}}{\text{TotalNumberofPrediction}}$$

- **F1-score**: is a statistical measure used to evaluate the performance of a classification model, particularly in situations where you have uneven class distributions or when you value precision and recall equally. It is the harmonic mean of precision and recall, providing a balance between them. The formula for the F1 score is:

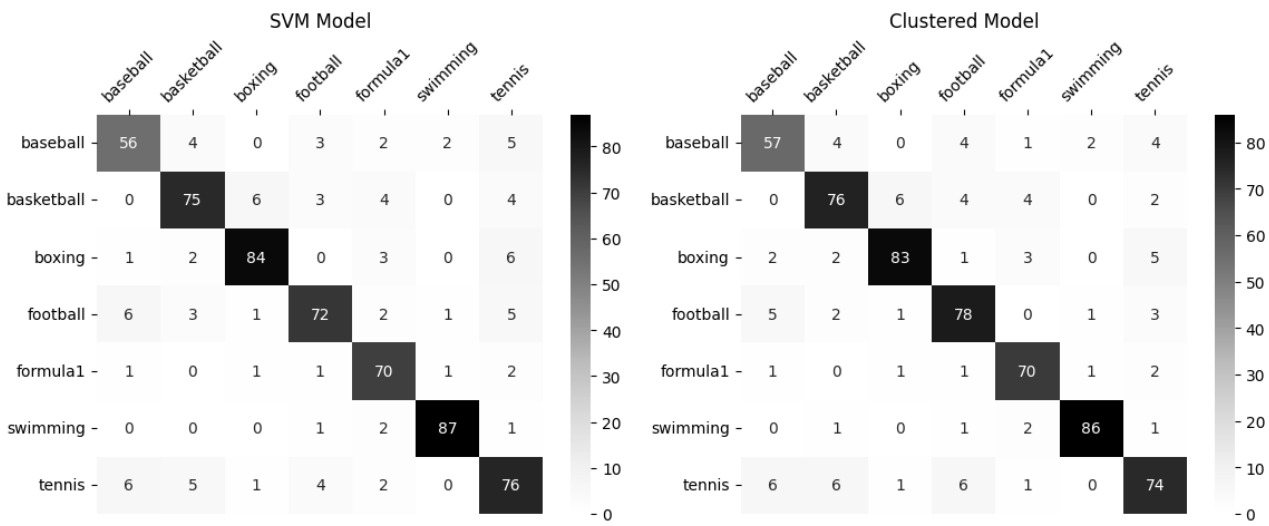
$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

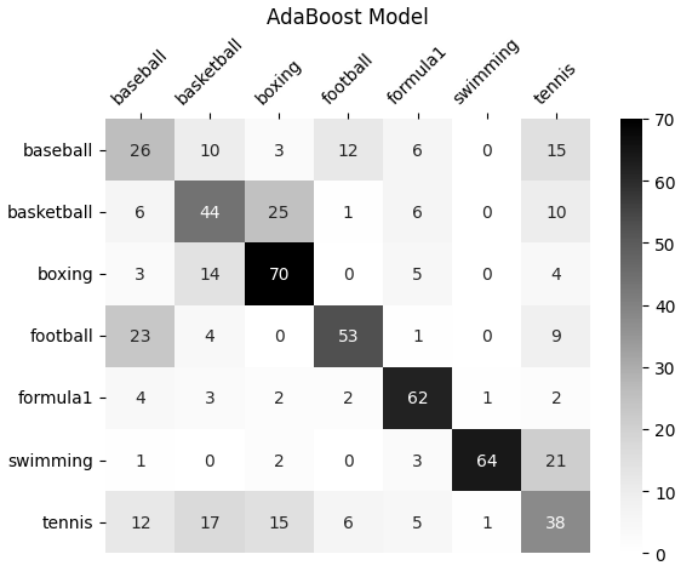
Results :

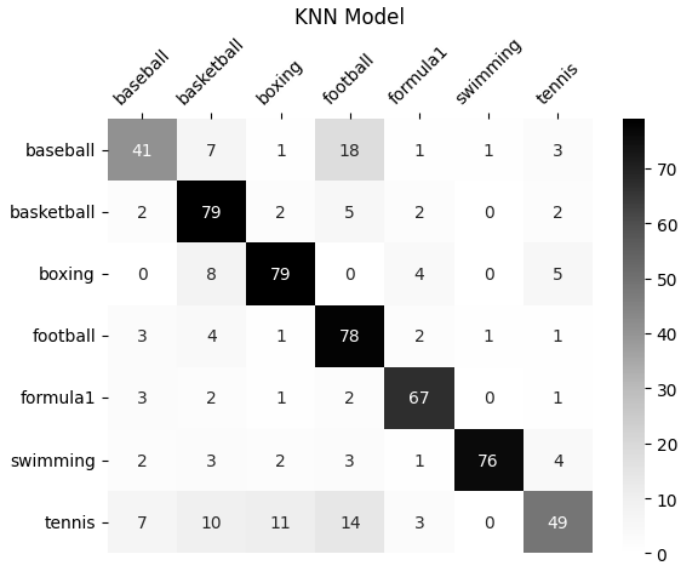
********** Linear SVM **********
Accuracy: 0.8346972176759411
F1-Score: 0.8339149537622009

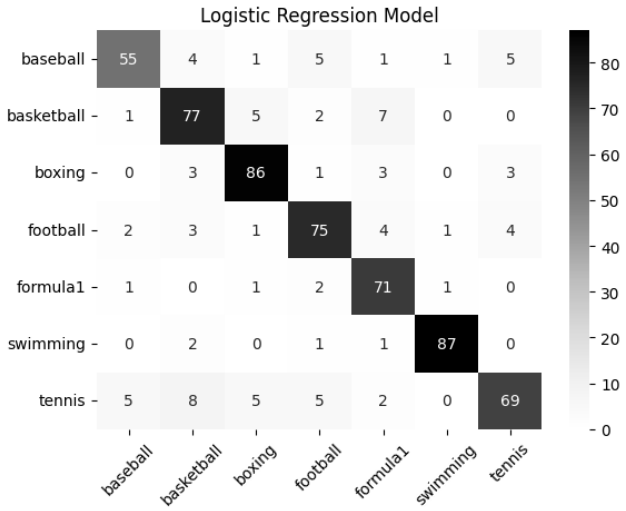
********** Clustered Model **********
Accuracy: 0.8297872340425532
F1-Score: 0.8289896319228429
********** AdaBoost Model **********
Accuracy: 0.5842880523731587
F1-Score: 0.5694586879834873
********** KNN Model **********
Accuracy: 0.7545008183306056
F1-Score: 0.752927444621216
********** Logistic Regression Model **********
Accuracy: 0.851063829787234
F1-Score: 0.85020172404558

4.2. Is there any confusion between the sports photos?



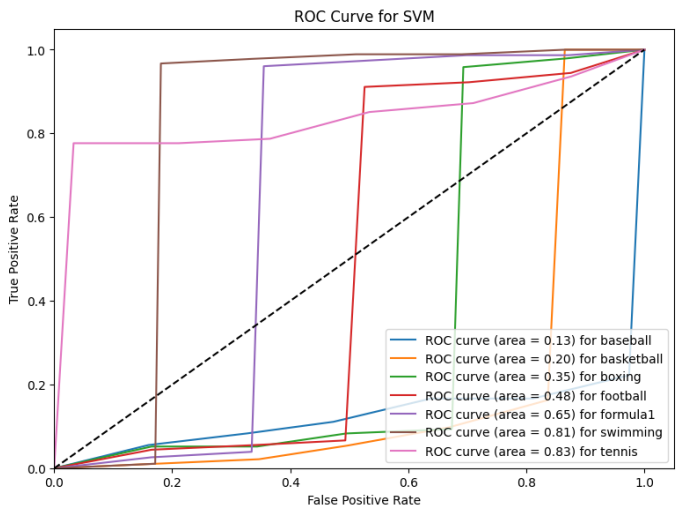


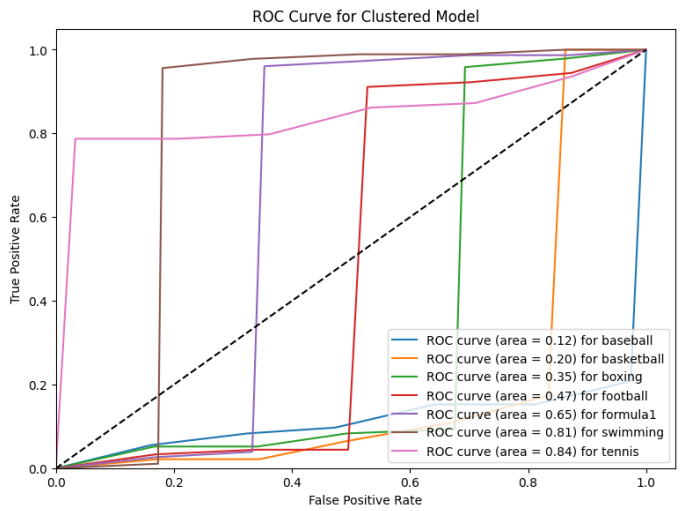


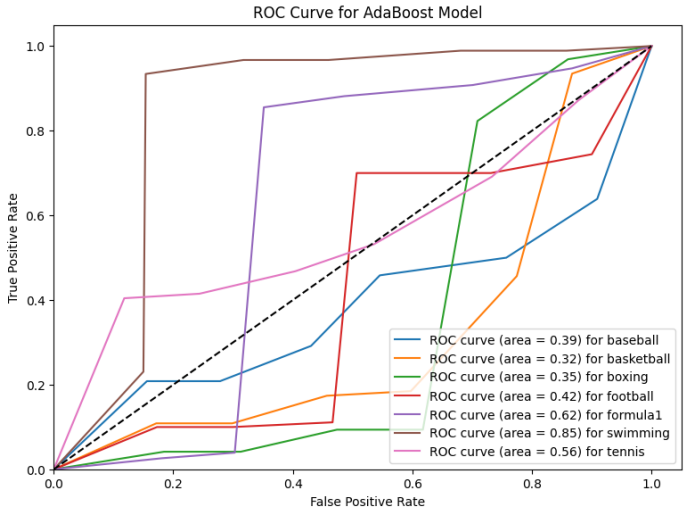


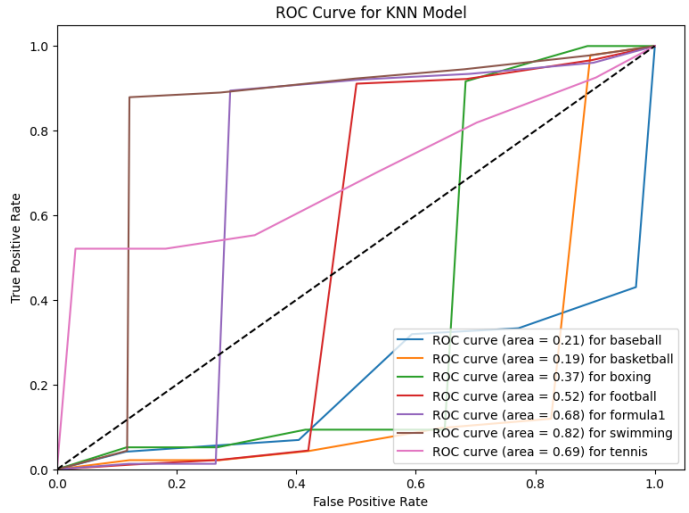
The Y-axis (vertical) represents the true classes of the samples (the actual labels), while the X-axis (horizontal) shows the predictions made by the model. Each cell of the matrix shows the number of samples for a given combination of actual class and predicted class.
The values on the main diagonal (top left to bottom right) indicate the number of samples correctly classified for each class. Values off the diagonal show classification errors.

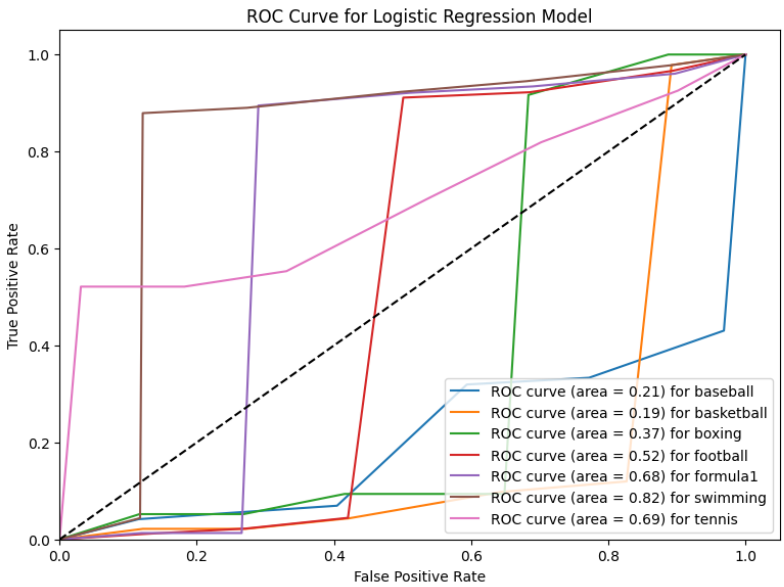
4.3. Which sports are easy or difficult to distinguish between?





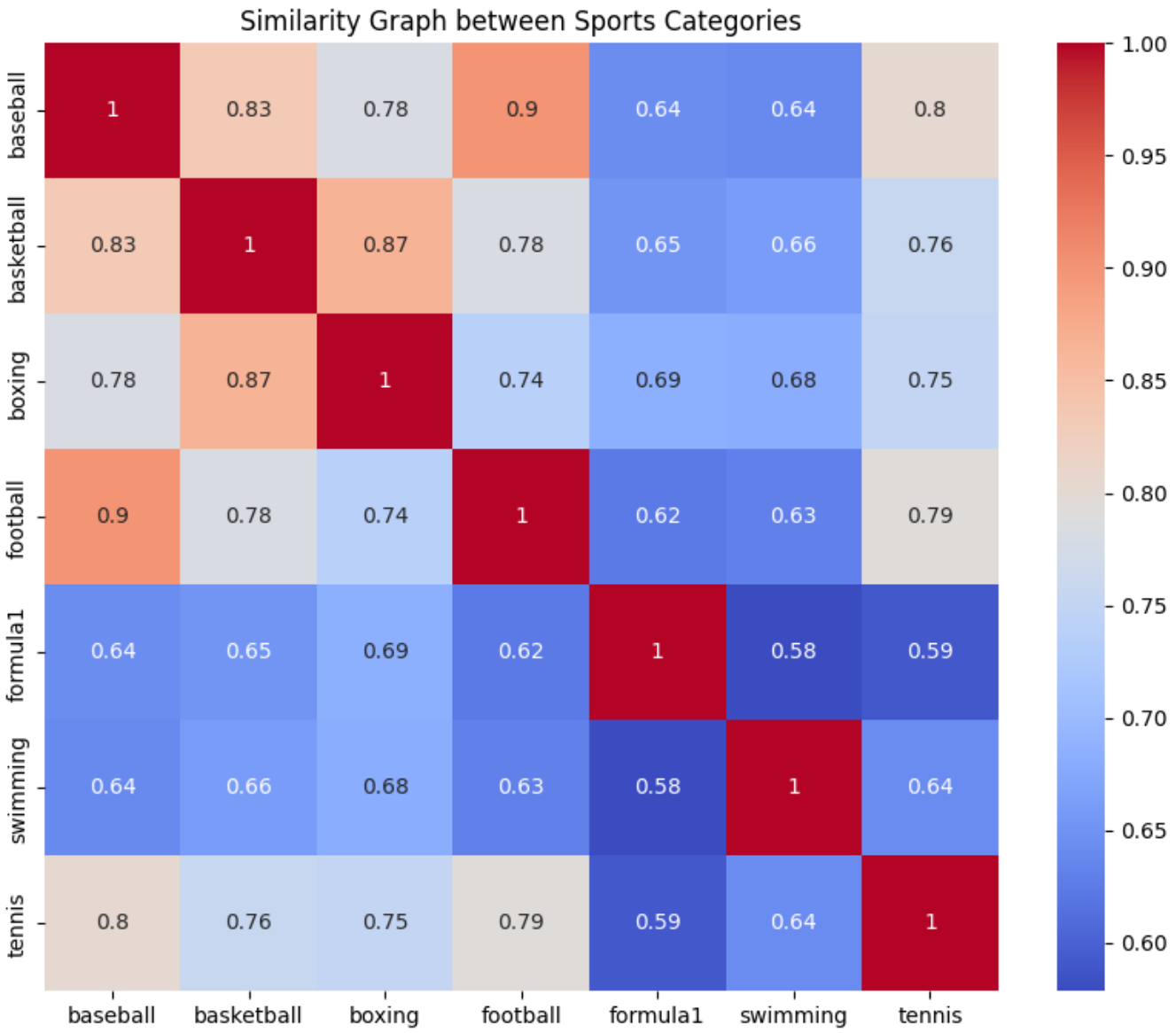






Used to evaluate the performance of a classification model.
X-axis (X): False Positive Rate, shows the proportion of negative examples incorrectly classified as positive. The lower this rate, the better.
Y axis (Y): Rate of True Positives, this axis shows the proportion of positive examples correctly identified. The higher this rate, the better.
AUC is an overall indicator of model performance. In general, a higher AUC indicates a better model.
We can see that the Clustered Linear SVM Model is the best one and tennis category demonstrates the highest level of prediction accuracy, whereas baseball category exhibits the lowest accuracy among the classes.

4.4. Are there groups that are more similar to each other?



In this graph, The more you get closer to 1, the more similarity there is between 2 sports. We can see that football and baseball are the most similar with a coefficient of 0.9, swimming an formula1 the least with a coefficient of 0.58.

5.Challenges

The challenges we faced in the project were with analyzing the images and their pixels. When the images had an average resolution of 476x322 pixels, it took a considerable amount of time to compute the results, and we did not receive any results at all. So we decided to reduce the resolution to 128x128 and we were able to obtain results, but they were so bad (like less than 50%), and in a second step we decided to reduce again to 64x64 and results were a little bit better (like 55%). After a long week of work, we concluded that we need to make progress on the resolution 128x128. So with the help of some adjustments (zoom, contrast on the images) and normalizatia, our program worked successfully and now have a success of 85-90%.

6. Conclusion

Upon examining the graphical representations, it becomes clear that the SVM model (normal and Clustered Linear SVM) stands out for its exceptional performance in comparison to the alternative models presented. This observation leads to the inference that conventional machine learning techniques may find it challenging to fully grasp the complex nature of the dataset at hand, resulting in less than optimal performance. Given this context, it's prudent to suggest a pivot towards the utilization of more advanced deep learning methodologies, specifically the adoption of Neural Network models. Such a shift is anticipated to significantly boost the model's ability to accurately interpret and analyze the data, thereby markedly improving overall model performance.