Project 1 – Supervised

Toma Sabin-Sebastian – Group 412

Image classification is a fundamental task in computer vision with applications ranging from medical diagnostics to autonomous driving. In this project, I explored different supervised learning approaches for classifying images. Our dataset is split into **80% training, 10% validation, and 10% testing,** ensuring a robust evaluation framework.

I implemented and compared multiple methods for feature extraction and classification, first one of them being **CNN and XGBoost**. **Convolutional Neural Networks (CNNs)** are used for **feature extraction**, leveraging their ability to capture spatial patterns in images. The extracted features are then passed to **XGBoost**, a powerful gradient boosting algorithm, for classification. For the second method, I used he **Histogram of Oriented Gradients (HOG)** method for feature extraction, which is widely used in traditional computer vision tasks and a **Decision Tree classifier** is trained on the extracted features to make predictions. To benchmark my methods against existing approaches, I also evaluated a **full CNN model**, where the CNN performs both **feature extraction and classification**.

To assess model performance, I evaluated predictions on the test set using **confusion matrices** and **Hamming Distance** for each approach. This comparative analysis provides insights into the effectiveness of combining deep learning-based and traditional machine learning-based methods for image classification.

In this documentation I will talk about the two following approaches:

1. XGBoost and CNN
2. Decision Tree and Histogram of Oriented Gradients(HOG)

XGBoost – CNN method

“**XGBoost** belongs to the family of boosting algorithms, which are ensemble learning techniques that combine the predictions of multiple weak learners”. It builds decision trees sequentially, just like other boosting methods, each new tree corrects errors made by previous trees and the final model is an **ensemble of weak learners (decision trees)** combined to make a strong predictor.

“**Convolutional Neural Networks (CNNs)** are a specialized class of neural networks designed to process grid-like data, such as images. They are particularly well-suited for image recognition and processing tasks. They are inspired by the visual processing mechanisms in the human brain, CNNs excel at capturing hierarchical patterns and spatial dependencies within images”. Unlike traditional neural networks, which treat images as **flat vectors,** CNNs **preserve spatial relationships** by using **convolutional layers** to **detect patterns and features** (edges, textures, objects).

Key components of a Convolutional Neural Network:

1. “**[Convolutional Layers](https://www.geeksforgeeks.org/what-are-convolution-layers/" \t "_blank):** These layers apply convolutional operations to input images, using filters (also known as kernels) to detect features such as edges, textures, and more complex patterns. Convolutional operations help preserve the spatial relationships between pixels.
2. [**Pooling Layers**](https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/)**:** They downsample the spatial dimensions of the input, reducing the computational complexity and the number of parameters in the network. Max pooling is a common pooling operation, selecting the maximum value from a group of neighboring pixels.
3. [**Activation Functions**](https://www.geeksforgeeks.org/activation-functions/)**:** They introduce non-linearity to the model, allowing it to learn more complex relationships in the data.
4. [**Fully Connected Layers**](https://www.geeksforgeeks.org/what-is-fully-connected-layer-in-deep-learning/)**:** These layers are responsible for making predictions based on the high-level features learned by the previous layers. They connect every neuron in one layer to every neuron in the next layer.”

How CNN work:

1. **Convolutional Layer (Conv Layer)** – Extracts features (e.g., edges, textures, shapes) using filters/kernels.

2. **Activation Function (ReLU)** – Introduces non-linearity to help the model learn complex patterns.

3**. Pooling Layer (Max/Average Pooling)** – Reduces dimensionality while keeping important features.

4**. Flattening** – Converts 2D feature maps into a 1D vector for input to the fully connected layer.

5**. Fully Connected Layer (Dense Layer)** – Makes final predictions based on extracted features.

6**. Softmax / Sigmoid Layer** – Outputs probability scores for classification.

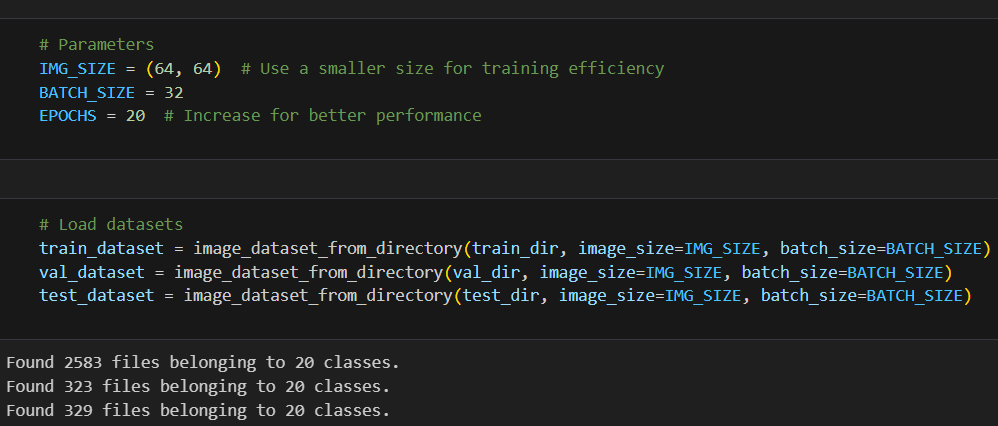


Figure 1. Parameters for base model and splitted datasets

Found 20 classes: ['Golf', 'bmw serie 1', 'chevrolet spark', 'chevroulet aveo', 'clio', 'duster', 'hyundai i10', 'hyundai tucson', 'logan', 'megane', 'mercedes class a', 'nemo citroen', 'octavia', 'picanto', 'polo', 'sandero', 'seat ibiza', 'symbol', 'toyota corolla', 'volkswagen tiguan']

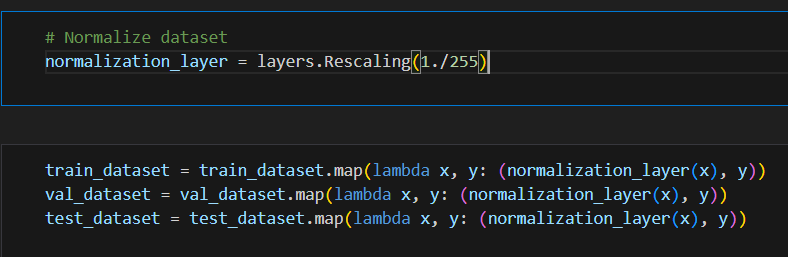


Figure 2. Normalizing the dataset

The pixel values scaled from [0,255] to [0,1] because neural networks perform better when the input values are normalized between 0 and 1. The x(images) is passed through normalization\_layer, transforming pixels values to [0,1] and the y(labels) remain unchanged.

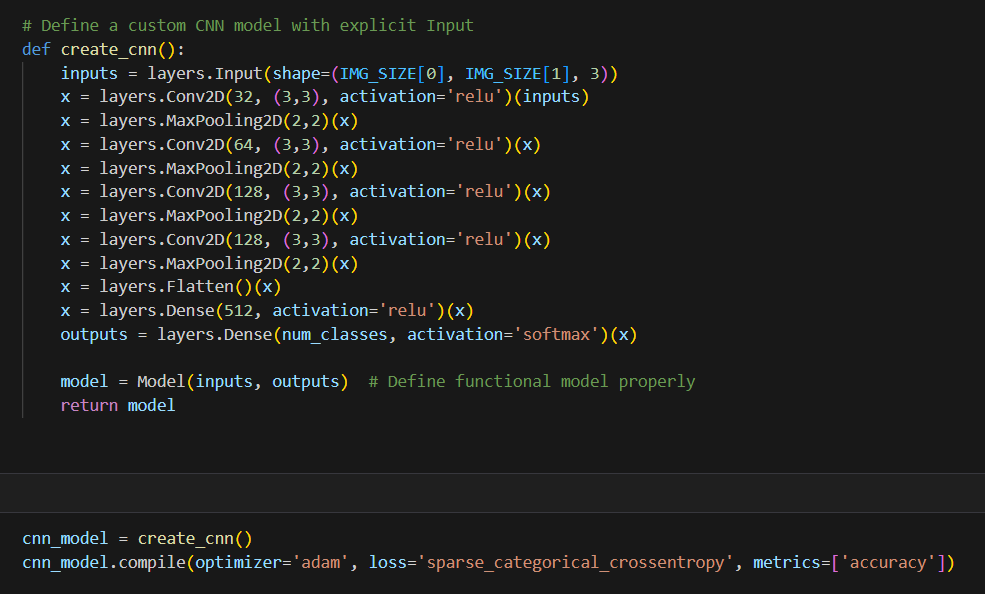


Figure 3. Custom CNN model

I created an input layer where IMG\_SIZE[0] and IMG\_SIZE[1] represent **image height and width** and 3 refers to **RGB channels** (color images). Then, I added **multiple Conv2D layers** to extract features (Conv2D(filters, kernel\_size, activation) → Extracts features like edges, textures; MaxPooling2D(pool\_size=(2,2)) → Reduces dimensions while keeping important features). After that, Flatten layer converts **2D feature maps** into a **1D vector** for classification, Dense layer (fully connected layer with 512 neurons) and final layer for multi-class classification. Finally, I compiled the model with: optimizer (adam), loss function (sparse\_categorical\_crossentropy) and metric (accuracy).

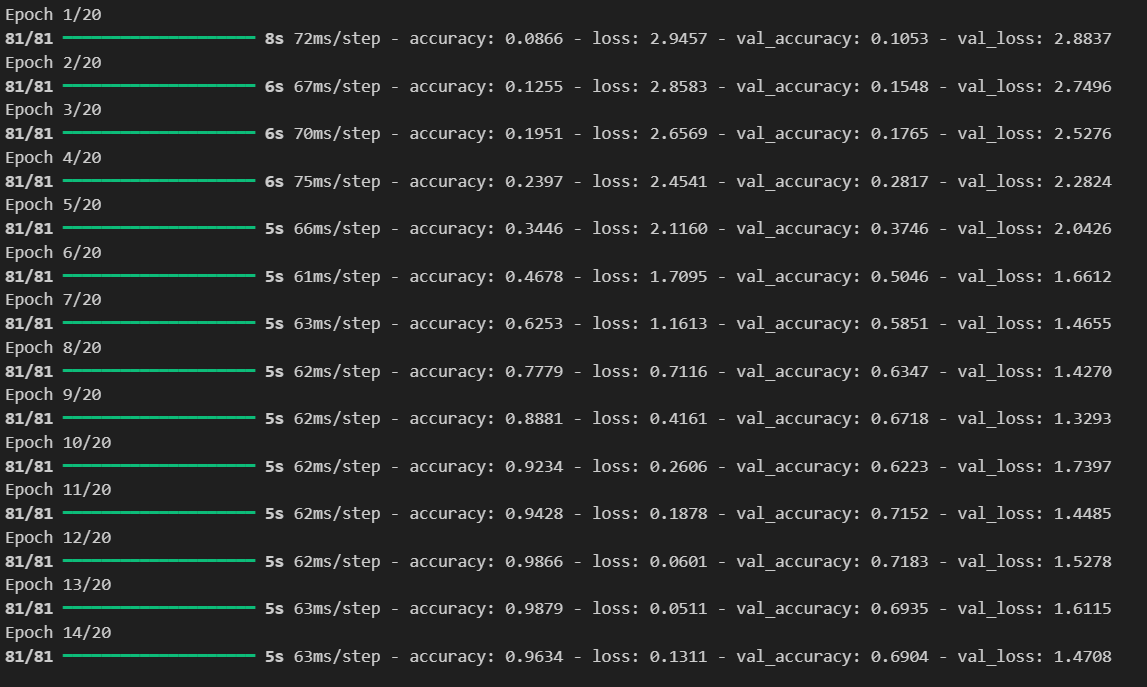
I tried 3 different optimizers: adam (Adaptive Moment Estimation), nadam (Nesterov-accelerated Adaptive Moment Estimation) and sgd (Stochastic Gradient descent).

Adam is an algorithm for optimization technique for gradient descent. Adam optimizer involves a combination of two gradient descent methodologies: **momentum** (keeps track of the average past gradients) and **adaptive learning** (adjusts learning rate based on squared past gradients).

Nadam **modifies Adam** by applying **Nesterov Accelerated Gradient (NAG)**, which **looks ahead** at the gradient before updating weights. Instead of computing the gradient at the current position, it **computes the gradient at the future estimated position.**

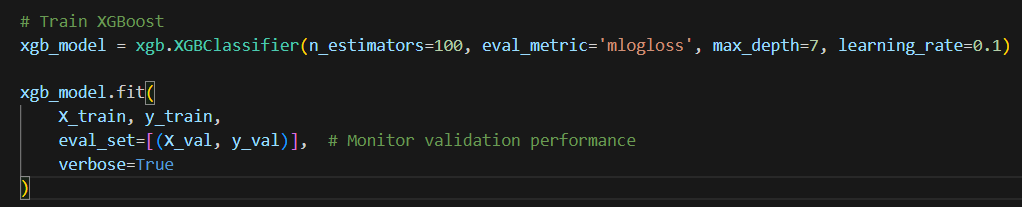
**SGD** updates weights **directly** based on the gradient of the loss function and unlike Adam and Nadam, it **does not adapt the learning rate**.

After compiling the CNN, I trained it with early stopping.

****

**Figure 4. Training log output of CNN model**

**Then, I used a CNN model to extract features from an intermiate layer, converted images into feature embeddings and prepared the extracted features for use in XGBoost model.**

****

**Figure 5. XGBoost model**

* n\_estimators defines **the number of trees** in the ensemble.
* **"mlogloss" (Multiclass Log Loss)** is used for **multi-class classification** and measures **how close** the predicted probabilities are to the true labels.

**I also used “merror” and “aucpr” for evaluation metric.**

**“Merror”** measures **misclassification rate** (how often the model **incorrectly** predicts the class), s**impler than** mlogloss, as it **only considers correctness**, not probability confidence.

“Aucpr” measures **Precision-Recall (PR) curve area**, used for **imbalanced datasets (Clio 344, Mercedes A-Klasse 84, Duster 280, BMW Serie 1 – 82).**

* **max\_depth** sets the **maximum depth of each decision tree**.
* learning\_rate controls **how much each tree contributes** to the final prediction.

After that I made the evaluation on test set.

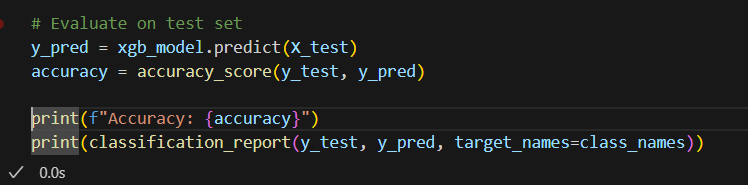


Figure 6. Evaluation on test set

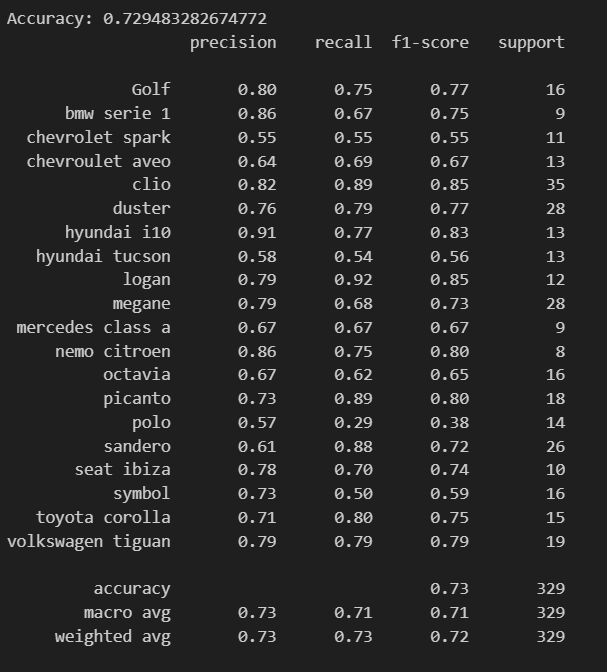


Figure 7. Classification report

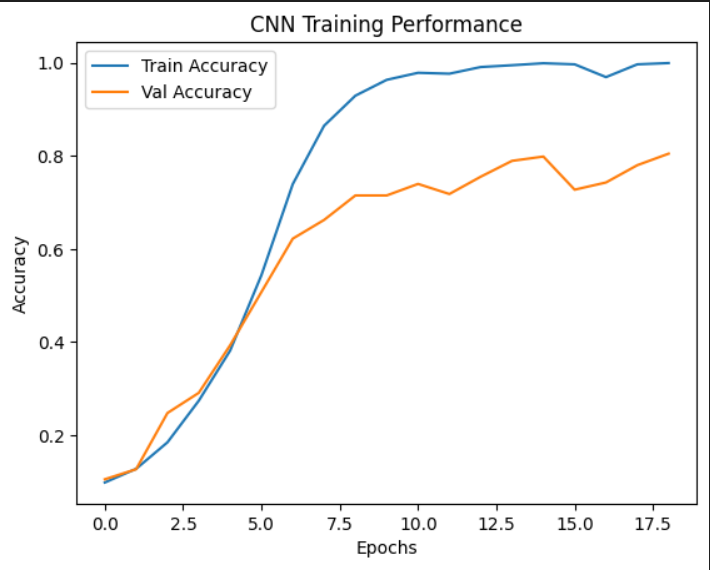


Figure 8. CNN training performance

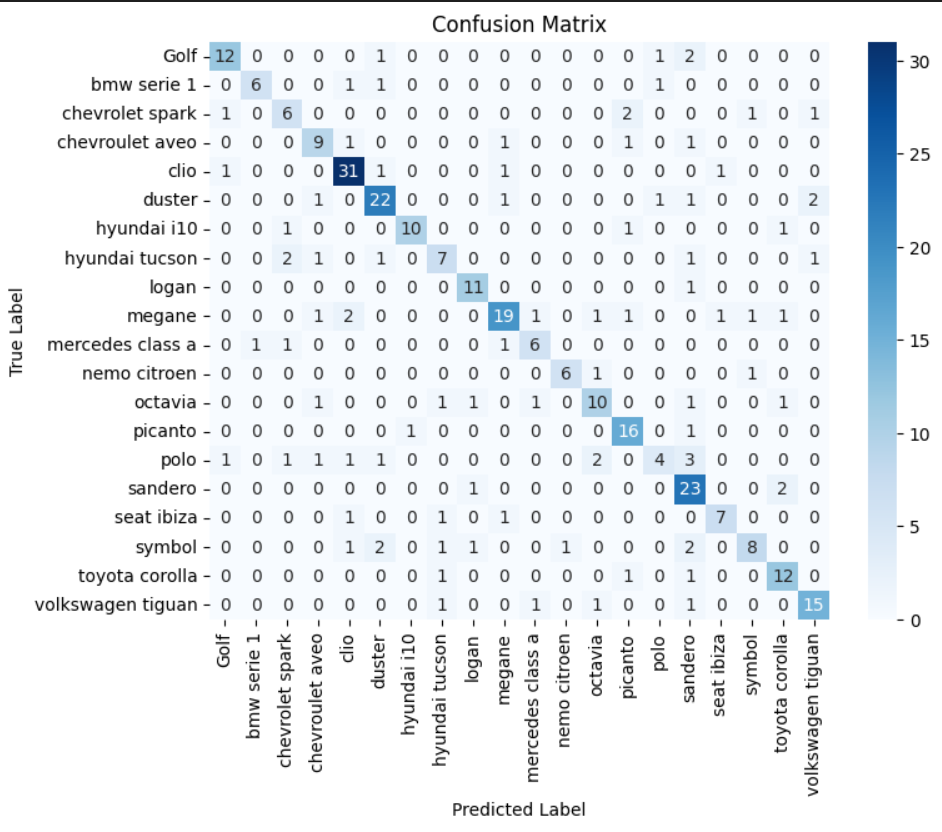


Figure 9. Confusion matrix

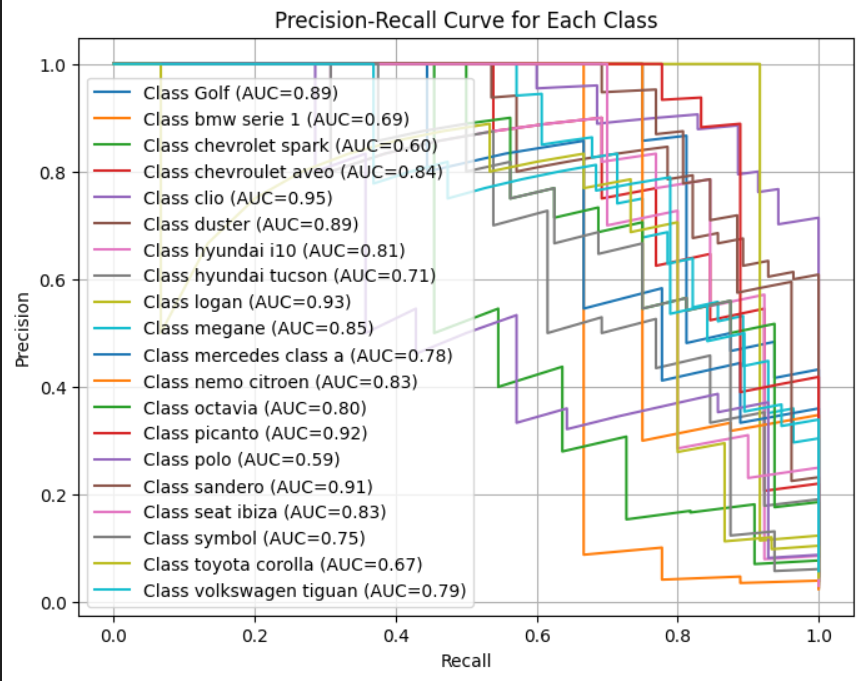


Figure 10. PR curve

**Precision (Y-axis)** → Measures how many of the positive predictions were actually correct. Higher is better.

Precision = True positives/True positives + False positives

**Recall (X-axis)** → Measures how many actual positive instances were correctly predicted. Higher is better.

Recall = True positives/True positives + False negatives

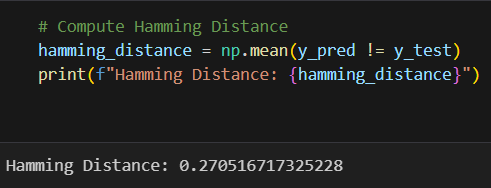


Figure 11. Hamming distance

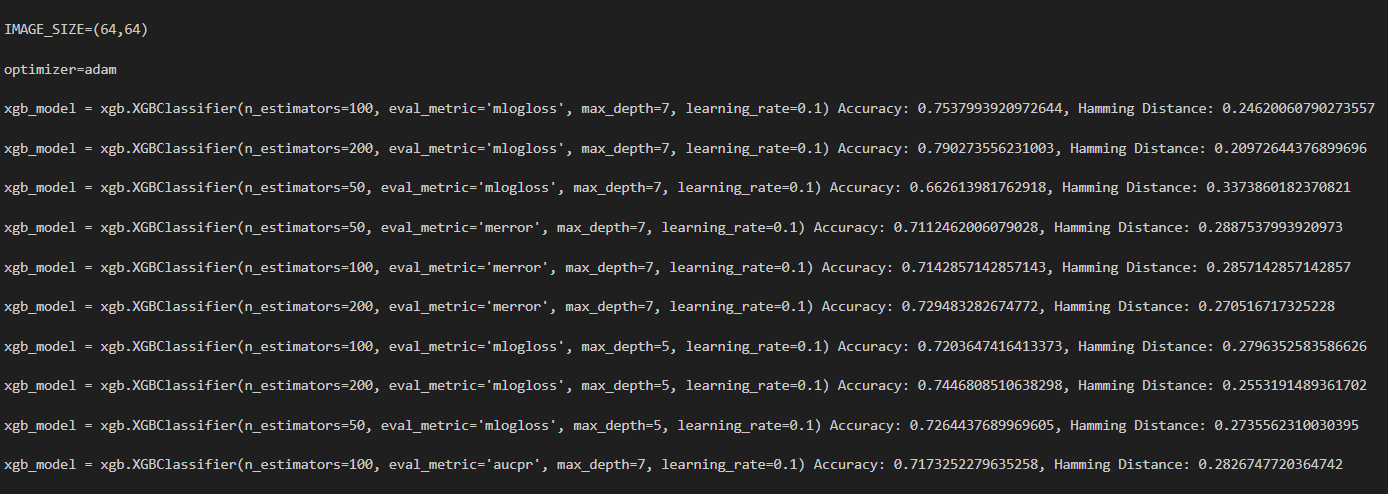


Figure 12. Hyperparameter tuning

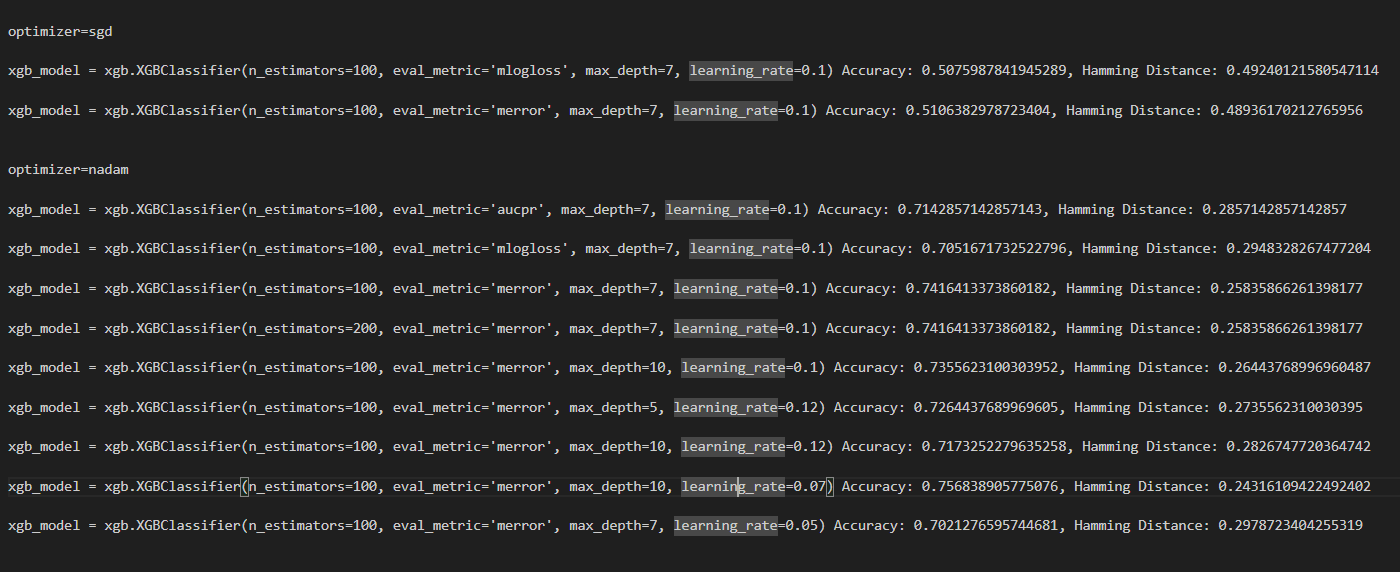


Figure 13. Hyperparameter tuning

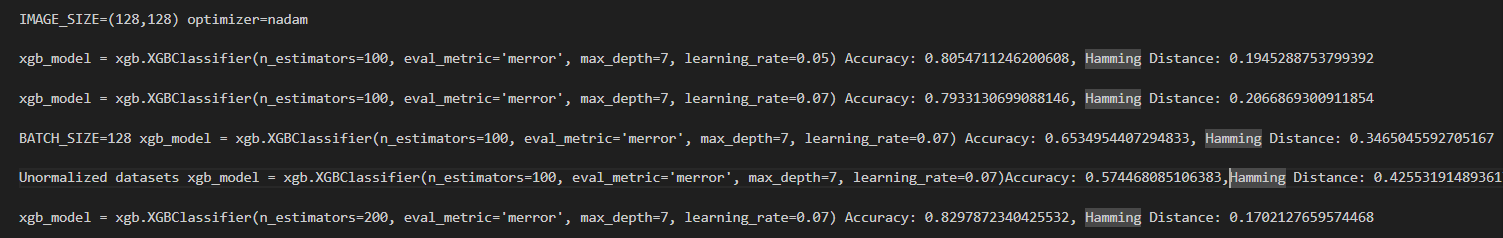


Figure 14. Hyperparameter tuning

Decision Tree - Histogram of Oriented Gradients(HOG) method

“Decision Tree Classifiers is a fundamental machine learning algorithm for classification tasks. They organize data into a tree-like structure where internal nodes represent decisions, branches represent outcomes and leaf node represent class labels.”

1. Root node is the starting point that represents the entire dataset.
2. **Branches** are the lines that connect nodes. It shows the flow from one decision to another.
3. **Internal Nodes**are Points where decisions are made based on the input features.
4. **Leaf Node** are the terminal nodes at the end of branches that represent final outcomes or predictions.

“In computer vision and image processing, the Histogram of Oriented Gradients (HOG) approach is used to explain the characteristics of a particular piece of data. It takes into account how frequently a particular area of a picture displays a gradient orientation.”

How HOG works:

1. **Gradient computation**: the image is converted to **grayscale** (if it is colored). Gradients (changes in intensity) are computed using **filters** in both horizontal and vertical directions that helps in capturing edge information.
2. **Cell and block division**: The image is divided into small regions called **cells (eg. 8x8 pixels).** A group of cells forms a **block (eg. 2x2 cells)** which is used for normalization.
3. For each cell, a histogram of gradient orientations (angles) is created. The histogram bins represent gradient angles (0°–180° or 0°–360°), and the values represent the magnitude of the gradients.
4. **Block normalization**: To handle illumination changes, the histogram values are normalized across larger overlapping blocks.
5. **Feature vector formation**: The final HOG descriptor is a concatenated vector of all normalized histograms from the image. This vector is then used as input for machine learning models like **Support Vector Machines (SVM)** for object detection.

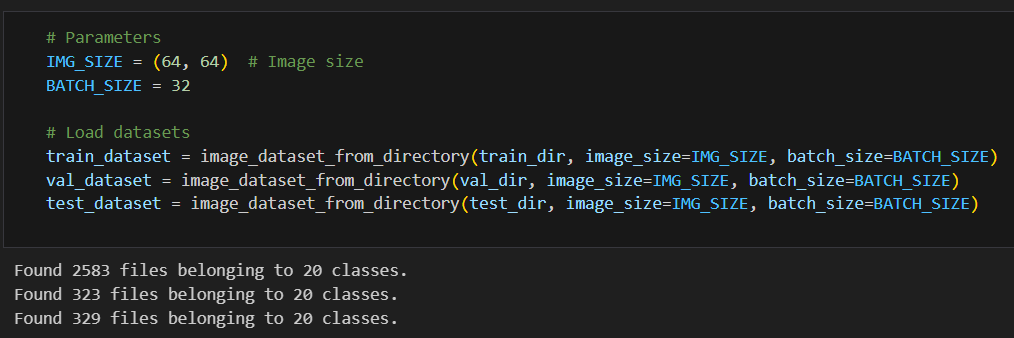


Figure 15. Parameters for base model and splitted datasets

Found 20 classes: ['Golf', 'bmw serie 1', 'chevrolet spark', 'chevroulet aveo', 'clio', 'duster', 'hyundai i10', 'hyundai tucson', 'logan', 'megane', 'mercedes class a', 'nemo citroen', 'octavia', 'picanto', 'polo', 'sandero', 'seat ibiza', 'symbol', 'toyota corolla', 'volkswagen tiguan']

After that, I extracted the features using HOG from images to datasets.

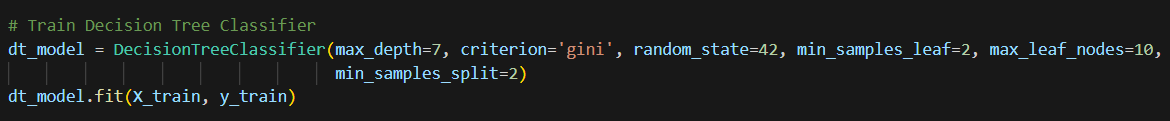


Figure 16. Decition tree classifier hyperparameters

* criterion: It measure **the quality of a split**. Supported values are 'gini', 'entropy' and 'log\_loss'. The default value is 'gini'
* max\_depth: This parameter denotes **maximum depth of the tree** (default=None).
* random\_state: Ensures **reproducibility by setting a fixed seed** for random splits.
* min\_samples\_leaf: The **minimum number of samples required to be at a leaf node** (default=1)
* max\_leaf\_nodes: It defines **the maximum number of possible leaf nodes**.
* min\_samples\_split: It defines the minimum number of samples required to split an internal node (default=2).

Gini Impurity measures how often a randomly chosen element from a set would be incorrectly labeled if randomly classified according to the class distribution in the set.

G=1−∑pi2​, pi – probability of class i, if a node is **pure** (contains only one class), Gini impurity is **0**.

Entropy measures the impurity of a node based on information theory. It quantifies the amount of uncertainty in the classification.

H=−∑pi​log2​(pi​), if a node is **pure**, entropy is **0**.

Then, I made the evaluation on test set.

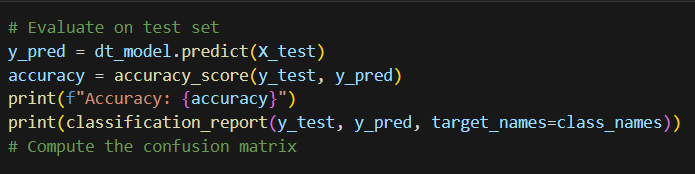


Figure 17. Evaluation on test set

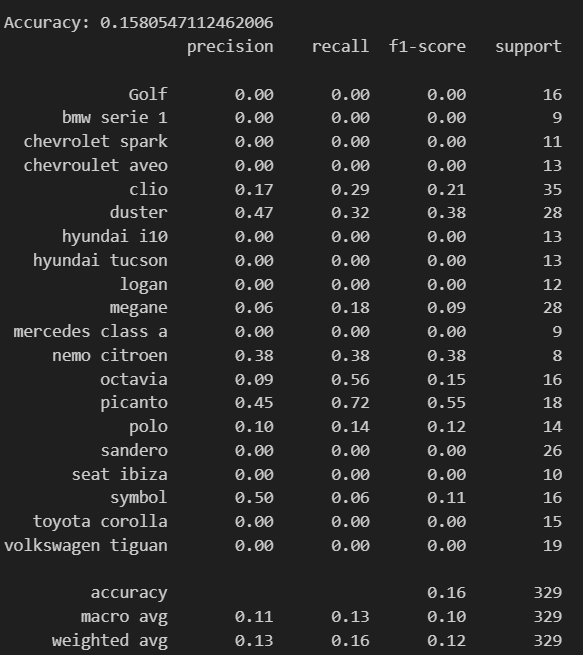


Figure 18. Classification report

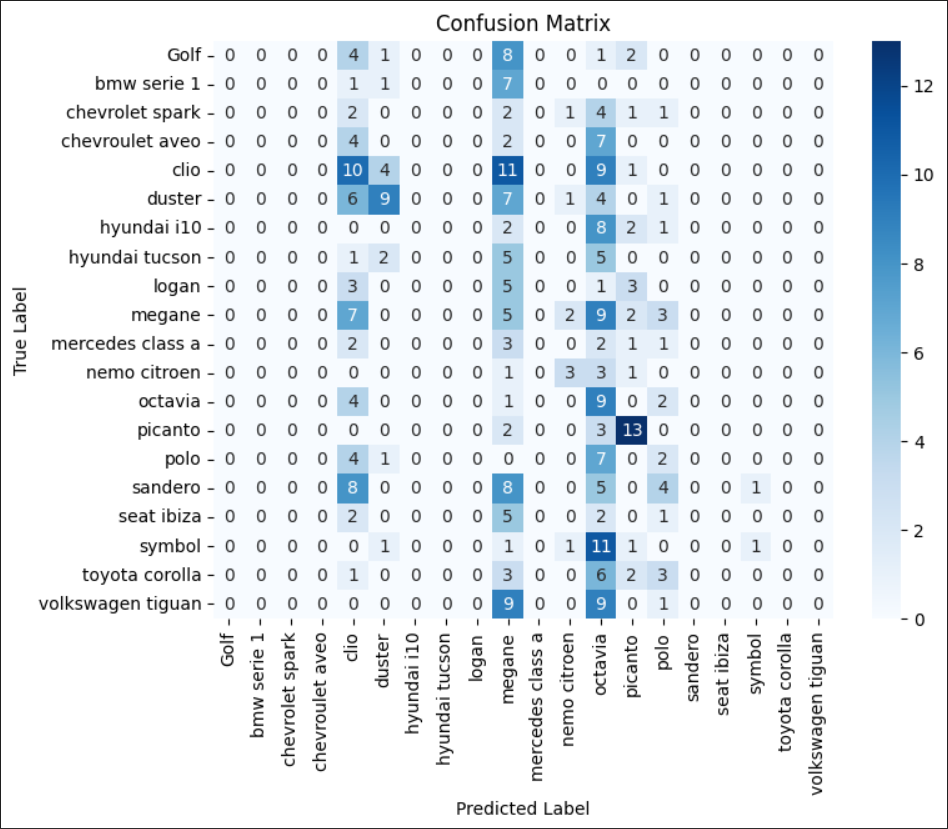


Figure 19. Confusion matrix

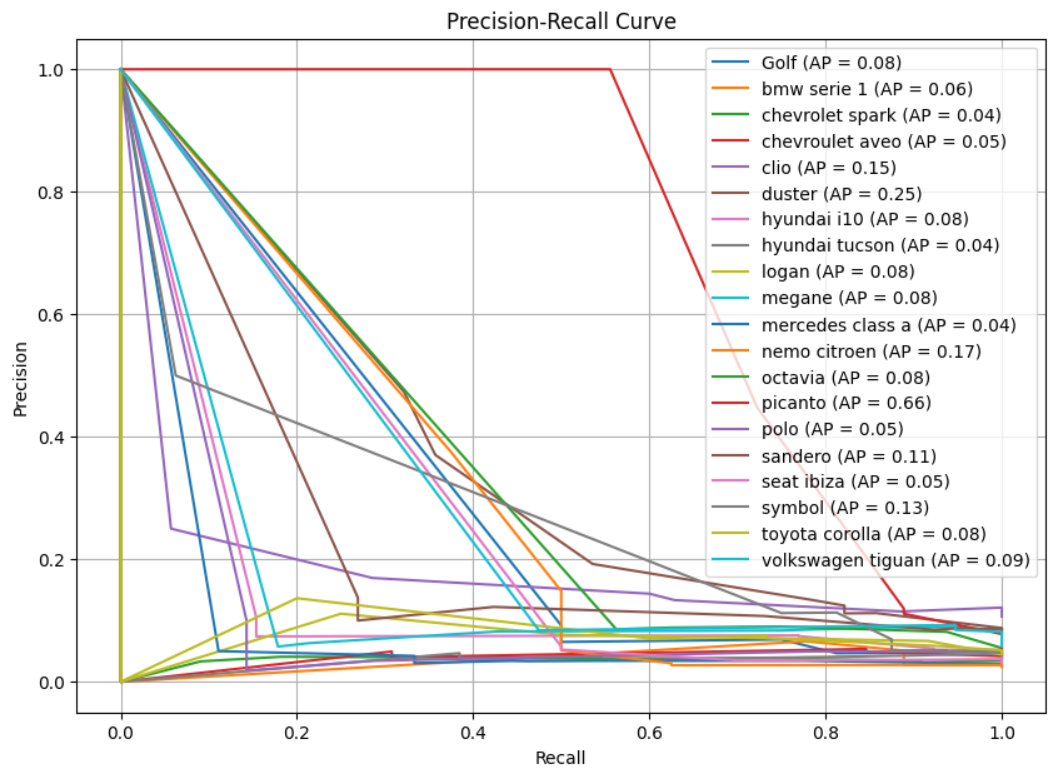


Figure 20. PR curve

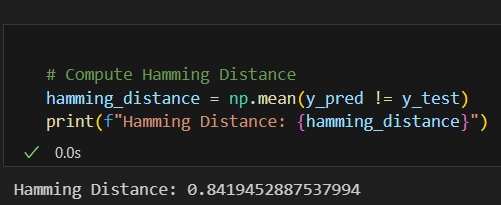


Figure 21. Hamming distance

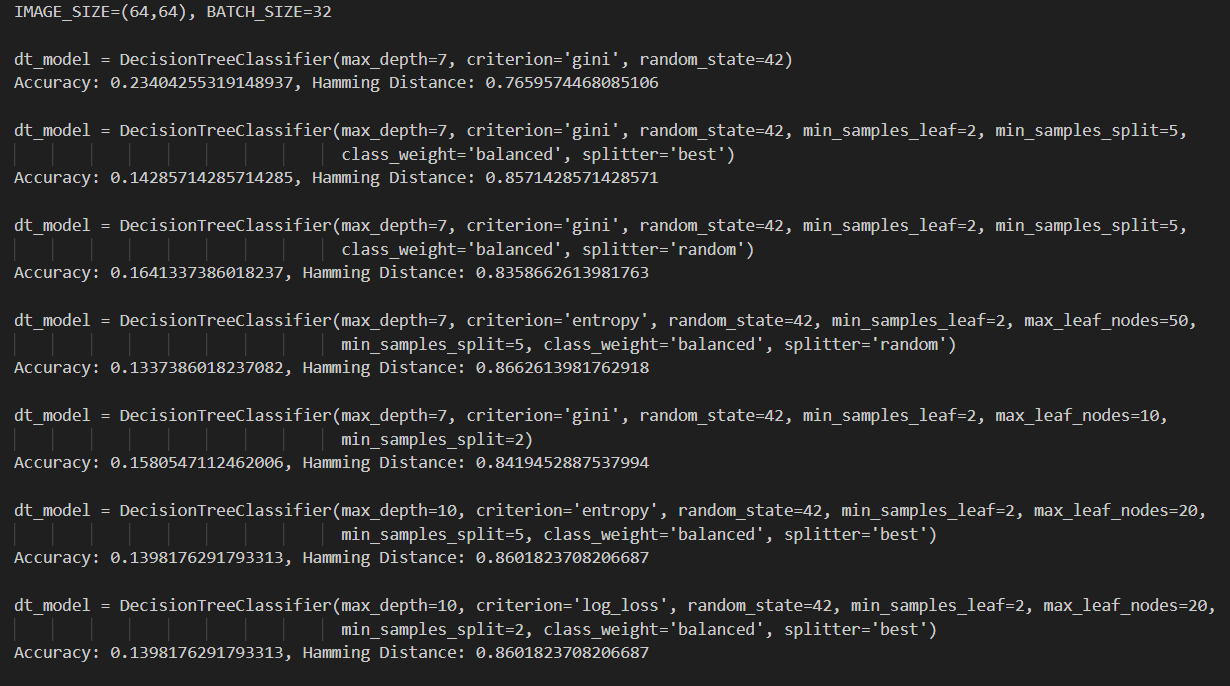


Figure 22. Hyperparameter tuning

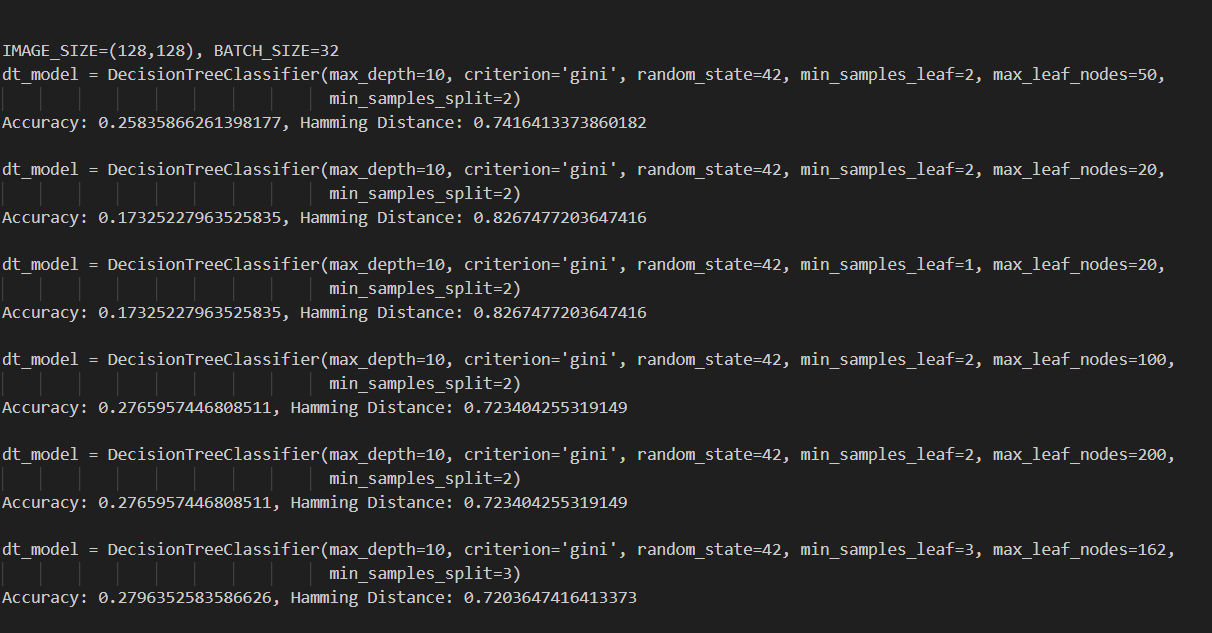


Figure 23. Hyperparameter tuning

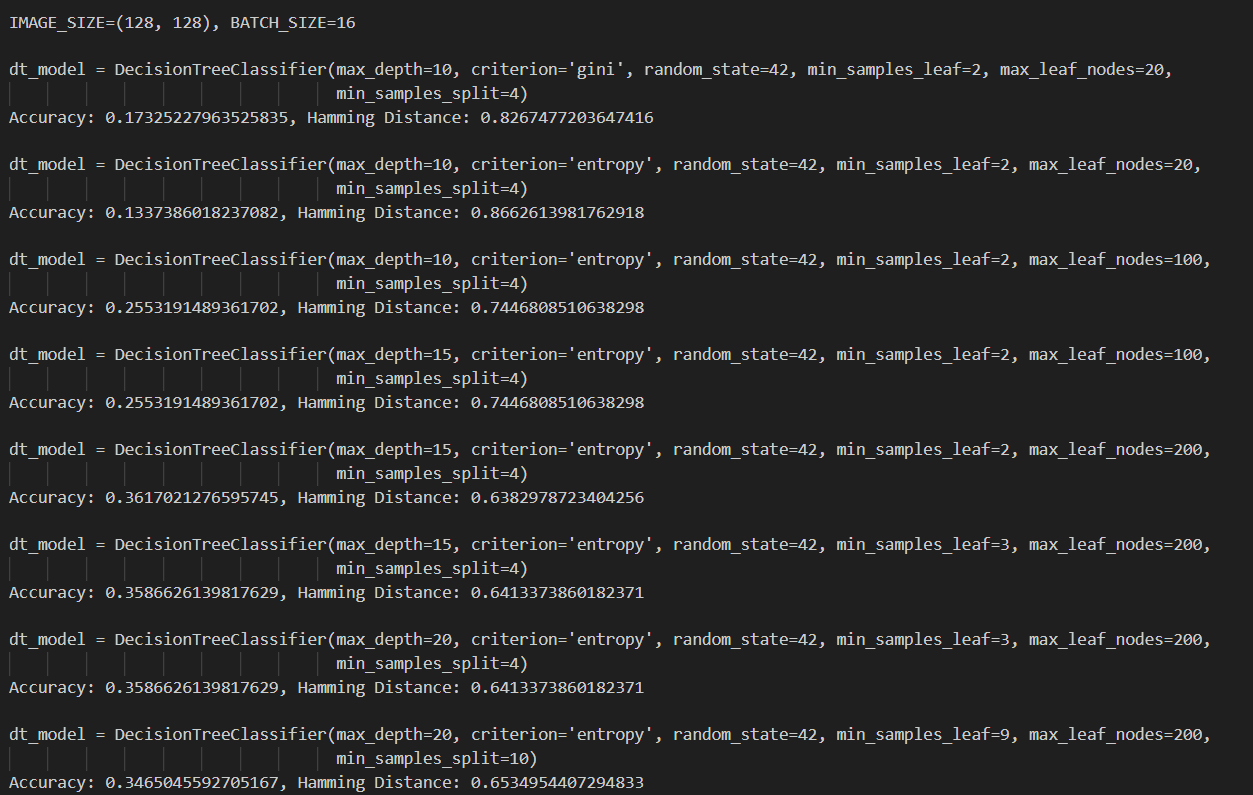


Figure 24. Hyperparameter tuning

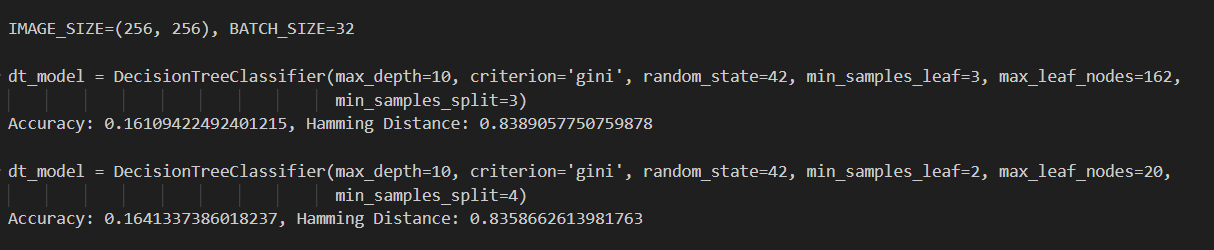


Figure 25. Hyperparameter tuning

Comparison with a method from literature

I evaluated a **full CNN model**, where the CNN performs both **feature extraction and classification**.

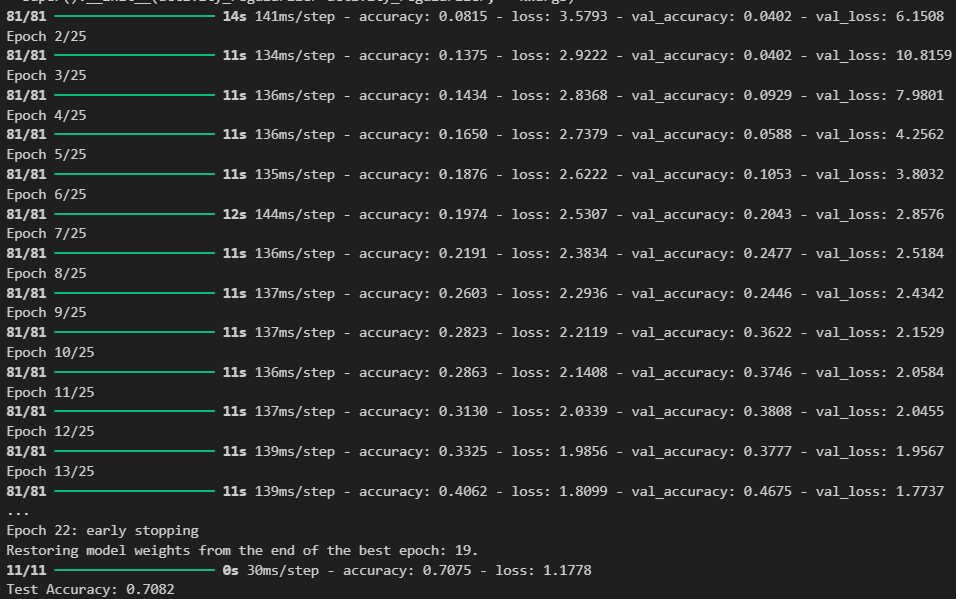


Figure 26. Training log output and accuracy CNN model

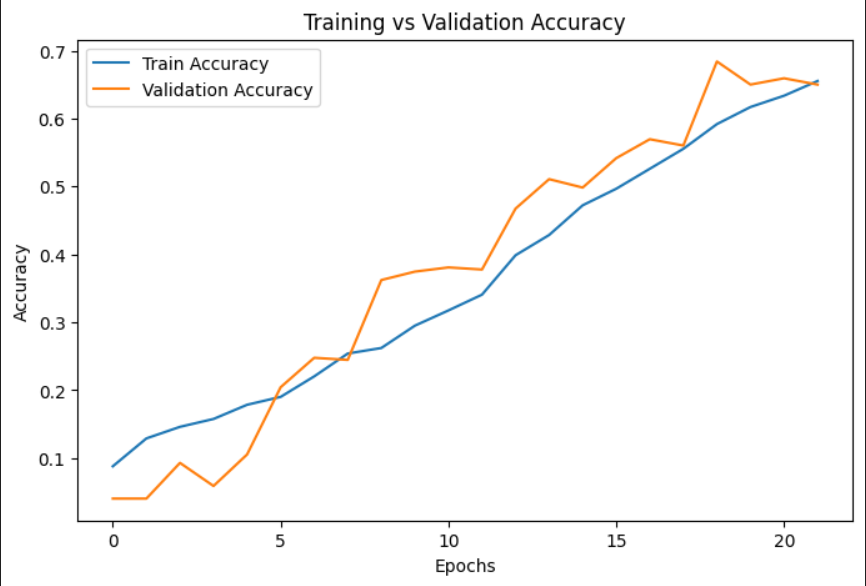


Figure 27. Training and validation accuracy

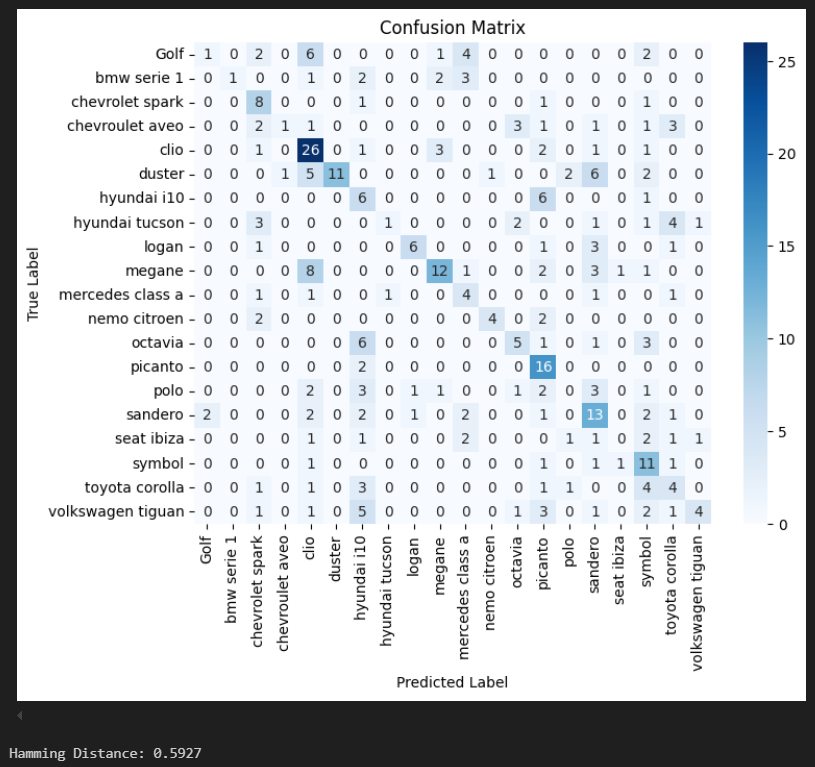


Figure 28. Confusion matrix and hamming distance

Bibliography:

1. [XGBoost - GeeksforGeeks](https://www.geeksforgeeks.org/xgboost/)
2. [Convolutional Neural Network (CNN) in Machine Learning - GeeksforGeeks](https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/)
3. [What is Adam Optimizer? - GeeksforGeeks](https://www.geeksforgeeks.org/adam-optimizer/)
4. [Decision Tree - GeeksforGeeks](https://www.geeksforgeeks.org/decision-tree/)
5. [Introduction to Histogram of Oriented Gradients (HOG) – Pythonista Planet](https://pythonistaplanet.com/histogram-of-oriented-gradients/)
6. [Building and Implementing Decision Tree Classifiers with Scikit-Learn: A Comprehensive Guide - GeeksforGeeks](https://www.geeksforgeeks.org/building-and-implementing-decision-tree-classifiers-with-scikit-learn-a-comprehensive-guide/)