Ahmed Tabşo **İbrahim Eltoma**

Demo Link :
https://colab.research.google.com/drive/1bXW8XjVmklsjYKcwvRieDGfSafnuyxlM#scrollTo=rfqEvSgvMo3J

Presentation Link:
https://www.canva.com/design/DAF4upiTvpw/SWamdDo89mXf_cf0QaMeRg/edit?utm_content=DAF4upiTvpw&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Classify The Traffic Signs Using Machine Learning Models

Introduction:

In our latest project, named "IA," we focused on making roads safer by teaching machines to recognize and understand traffic signs. Unlike the common use of methods like deep learning, we opted for simpler machine learning algorithms. This choice allowed us to explore various classic methods to find the best way to identify traffic signs accurately.

Our project started with getting a well-organized dataset that had labeled images of different traffic signs. This dataset became the building block for training our machine learning algorithms. The diverse collection of pictures in the dataset gave our models a solid foundation to learn and perform well.

To achieve our goal, we used a mix of different machine learning algorithms, such as 'K Neighbors Classifier,' 'MLP Classifier,' 'Decision Tree Classifier,' 'Random Forest Classifier,' 'Gradient Boosting Classifier,' 'Cat Boost Classifier,' and 'LGBM Classifier.' Each algorithm was chosen for specific reasons, and together they helped us understand the complexities of recognizing traffic signs.

While many people usually turn to deep learning for projects like ours, we decided to explore traditional machine learning methods. By avoiding deep learning, we aimed to show that we can still achieve excellent results by using simpler and more understandable techniques.

In this article, we share our journey, discussing the experiences and challenges we faced. Most importantly, we reveal the outcomes of using different machine learning methods for recognizing traffic signs. Join us as we explore the intersection of technology and transportation, working towards smarter and safer roads.

To kick off our project, we began by bringing in the dataset and connecting it to our work. This dataset contained images of traffic signs, each with a size of 32 x 32 pixels. In total, there were 4298 images, all neatly divided into 19 different categories.

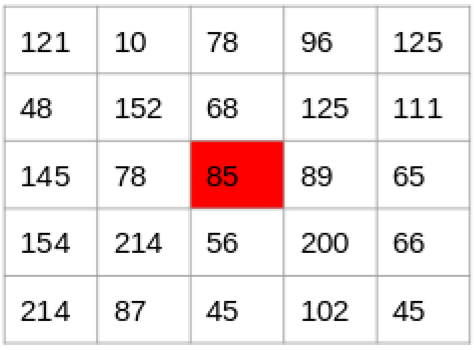
Next, we took these images and worked on extracting features from them. After that, we flattened the images, making them suitable for our models to work with.

Once our dataset was prepped, we split it into two parts: 80% for training our models and 20% for testing how well they learned. With this division, we ensured that our models had a good balance of learning and checking their performance.

Moving on to the next step, we introduced our models to the data for feature extraction. We measured their accuracy and found out how well they could understand and classify the traffic signs.

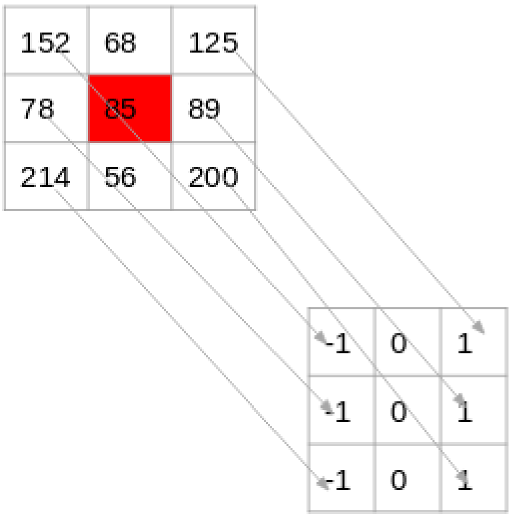
In our feature extraction process, we utilized the Prewitt method, a widely employed technique in the realm of image processing. This method is known for its simplicity and effectiveness in accentuating both vertical and horizontal edges within an image. By emphasizing these edges, we aim to capture significant features that contribute to the overall understanding of the image content. Prewitt's approach proves particularly valuable in computer vision tasks, providing a reliable means of detecting edges and extracting essential features for further analysis and interpretation.

Let’s say we have the following matrix for the image:



121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

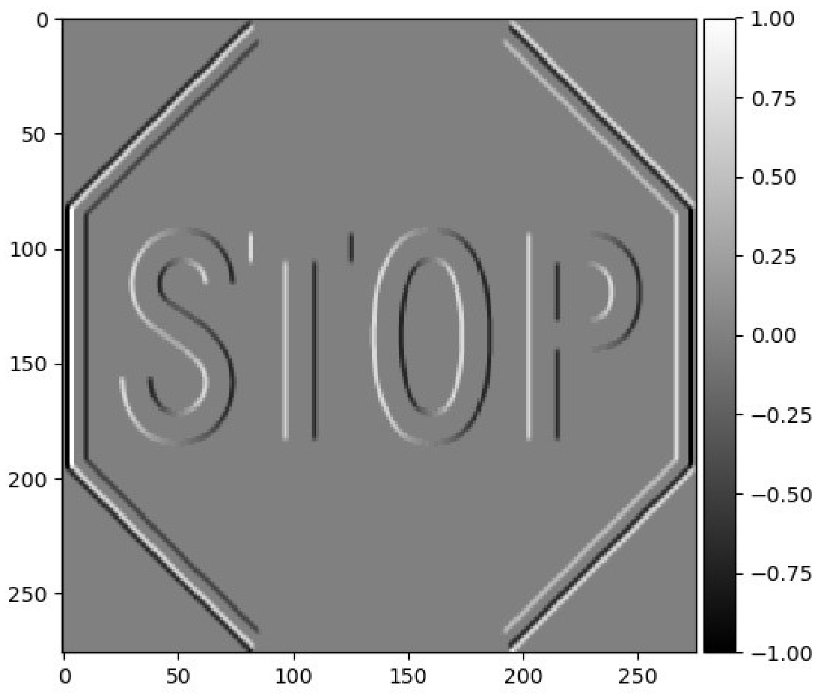
Now consider the pixel 85 highlighted in the below image:



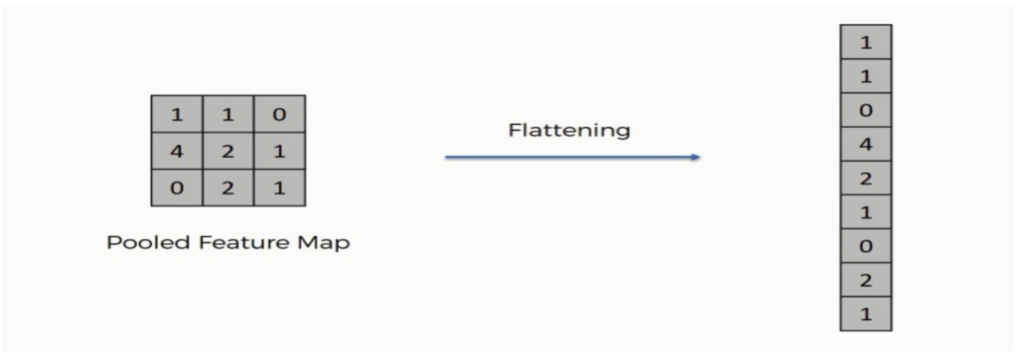
152	68	125
78	85	89
214	56	200

-1	0	1
-1	0	1
-1	0	1

after applying the Prewitt method on our project we got this result for feature extraction:



After Prewitt method to extracting the features we have to apply flattening operation to make the photos suitable to apply the machine learning algorithms.
Here it is the explanation of flattening:



Now, let's delve into the machine learning algorithms we used and their simplified explanations:

Suppose we have a dataset with two features (X1 and X2) and binary labels (y), and we want to use KNN for binary classification. Our dataset looks like this:K Neighbors Classifier (KNN):

The K Neighbors Classifier is a non-parametric algorithm used for classification and regression. In the context of classification, it assigns a data point to the class most common among its k nearest neighbors.
For example:

Suppose we have a dataset with two features (X1 and X2) and binary labels (y), and we want to use KNN for binary classification. Our dataset looks like this:

Instance	Feature 1 (X1)	Feature 2 (X2)	Label (y)
1	2	3	0
2	4	6	0
3	5	4	1
4	7	5	1

We want to classify a new instance with features (X1 = 6, X2 = 4).

Step 1: Choose the Number of Neighbors (k)

Let's say we choose
K=3.

Step 2: Compute Euclidean Distances

Compute the Euclidean distance between the new instance and each existing instance in the dataset.

$$\text{Distance to Instance 1} = \sqrt{(6-2)^2 + (4-3)^2} = \sqrt{16+1} = \sqrt{17}$$
$$\text{Distance to Instance 2} = \sqrt{(6-4)^2 + (4-6)^2} = \sqrt{4+4} = \sqrt{8}$$
$$\text{Distance to Instance 3} = \sqrt{(6-5)^2 + (4-4)^2} = \sqrt{1+0} = \sqrt{1}$$
$$\text{Distance to Instance 4} = \sqrt{(6-7)^2 + (4-5)^2} = \sqrt{1+1} = \sqrt{2}$$

Step 3: Select the Nearest Neighbors

Select the k=3 instances with the smallest distances. In this case, the nearest neighbors are Instances 3, 4, and 2.

Step 4: Make a Prediction

For classification, the algorithm typically takes a majority vote. If more neighbors belong to Class 1, the prediction is Class 1; otherwise, it's Class 0.

Instances 3 and 4 belong to Class 1.
Instance 2 belongs to Class 0.
Therefore, the majority vote is Class.

MLP Classifier (Multi-Layer Perceptron):
Explanation: MLP Classifier is like a smart brain with layers that work together to understand and categorize things. It's excellent for sorting stuff into different groups.

Decision Tree Classifier:
Imagine making decisions by asking a series of questions. Decision Tree Classifier does just that, creating a tree-like structure to help classify things based on answers to questions.

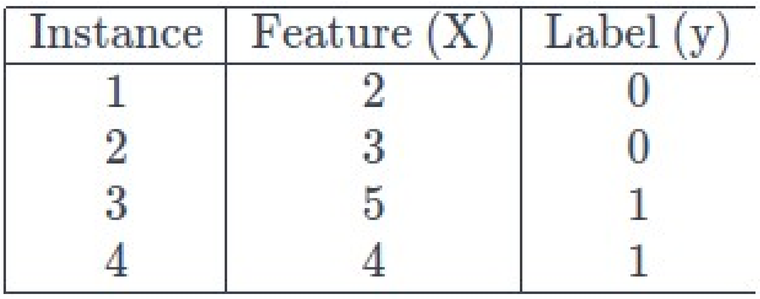
Random Forest Classifier:
Random Forest Classifier is like a team of Decision Tree Classifiers, each looking at a random piece of information. They vote on the best answer, making the final decision more robust.

Gradient Boosting Classifier:
This algorithm learns from its mistakes, like getting better after each attempt. It builds a series of smart learners, each fixing the errors of the previous ones.

Cat Boost Classifier:
Cat Boost Classifier is a cool tool for handling categories efficiently. It's like a pro at dealing with different types of things without needing extra preparation.

For example :

Suppose we have a dataset with one feature (X) and binary labels (y), and we want to use CatBoostClassifier for binary classification.



Instance	Feature (X)	Label (y)
1	2	0
2	3	0
3	5	1
4	4	1

Step 1: Initialize the Model

Initialize the model with a constant value. For binary classification, we often use the log-odds of the class probabilities.

$$F_0(x) = \ln\left(\frac{\Pr(Y=1)}{\Pr(Y=0)}\right)$$

In practice, we often use $F_0(x) = 0$, assuming equal class probabilities.

Step 2: Compute Pseudo-Residuals

Compute the pseudo-residuals, which represent the negative gradient of the loss function with respect to the current model's output. For binary classification with a logistic loss:

$$r_i^{(m)} = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} = y_i - p_i^{(m-1)}$$

Step 3: Fit a Weak Learner (Tree)

Fit a weak learner (decision tree) to the pseudo-residuals. Here, CatBoost automatically handles categorical features efficiently.

Step 4: Update the Model

Update the model by adding a fraction (α) of the weak learner's output.

$$F_m(x) = F_{m-1}(x) + \alpha \cdot h_m(x)$$

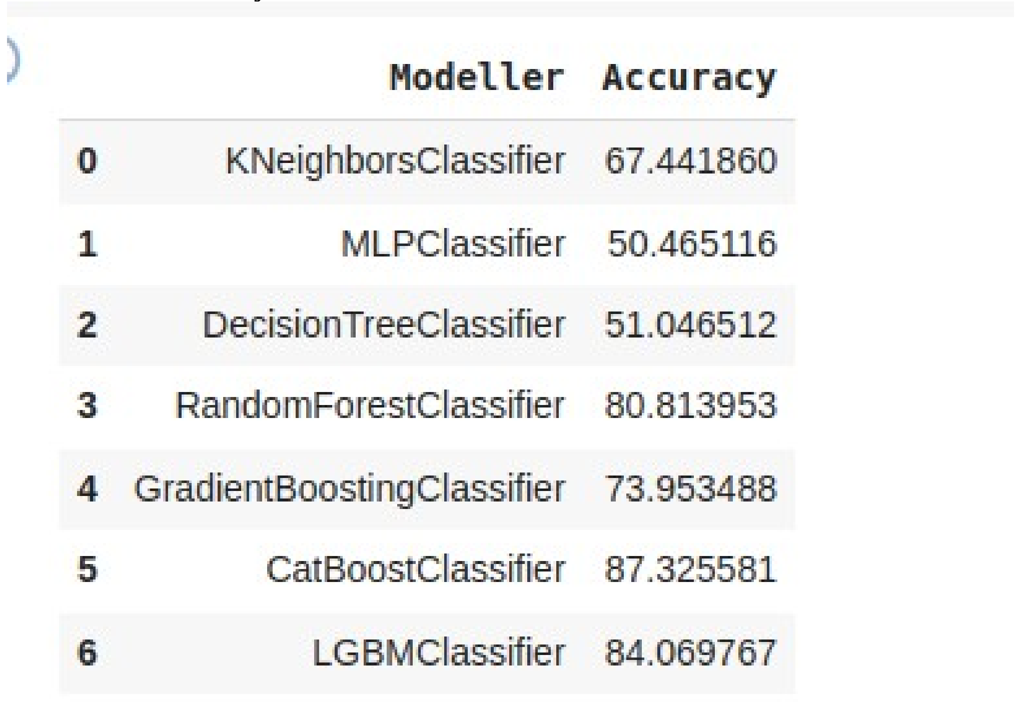
where $h_m(x)$ is the output of the weak learner.

Step 5: Repeat Steps 2-4

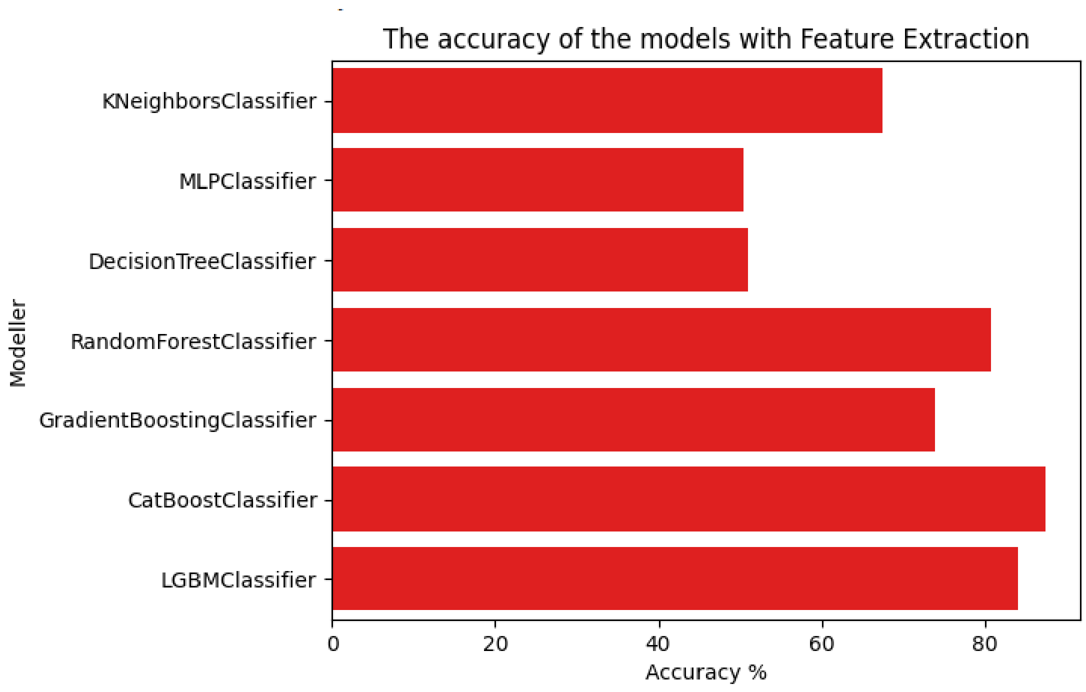
Repeat Steps 2-4 for a predefined number of iterations or until convergence.

LGBM Classifier (Light GBM):
LGBM Classifier is like a super-fast learner, great for dealing with lots of information quickly. It's perfect for handling big datasets with ease.

and this is the models accuracy :

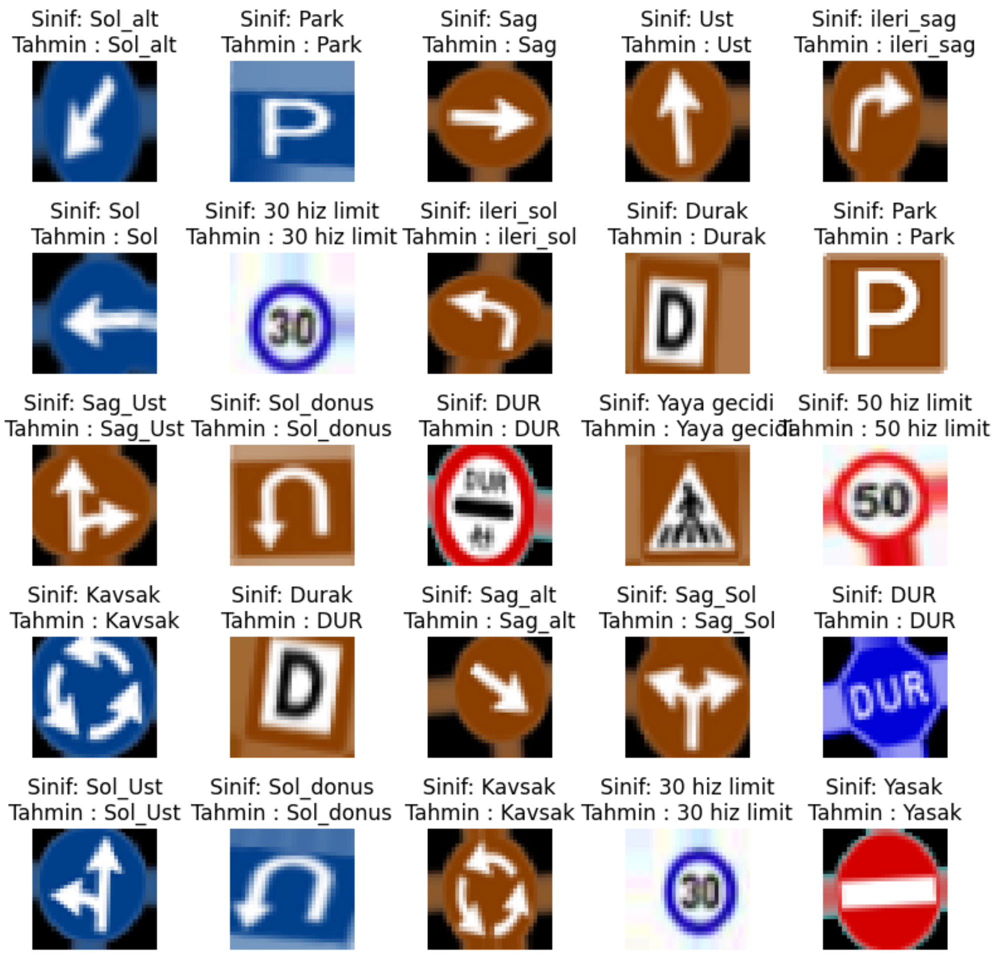


	Modeller	Accuracy
0	KNeighborsClassifier	67.441860
1	MLPClassifier	50.465116
2	DecisionTreeClassifier	51.046512
3	RandomForestClassifier	80.813953
4	GradientBoostingClassifier	73.953488
5	CatBoostClassifier	87.325581
6	LGBMClassifier	84.069767

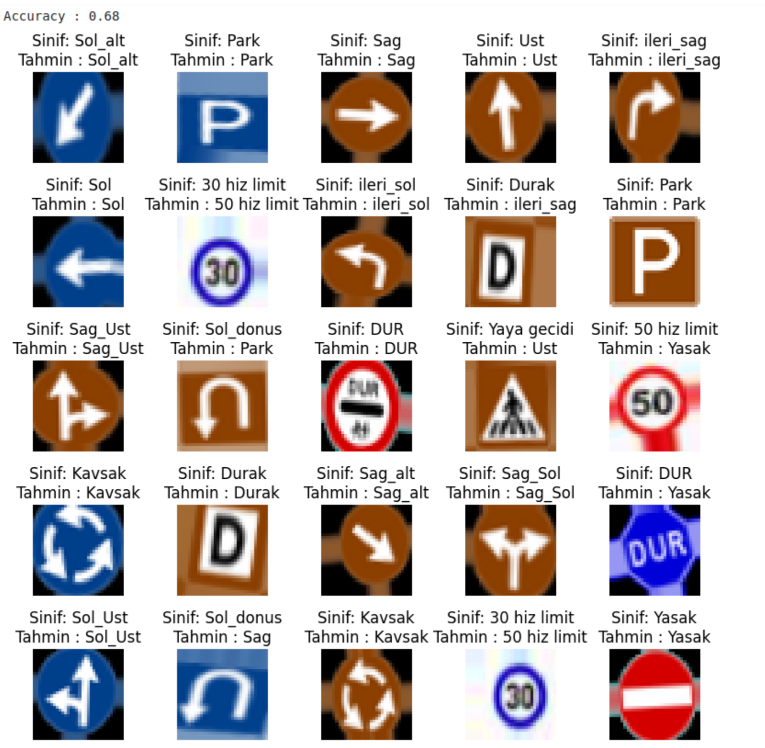


We did the features extraction with these algorithms and we got these results :
for the Cat Boost Classifier we got this result:

```
Accuracy : 0.96
```

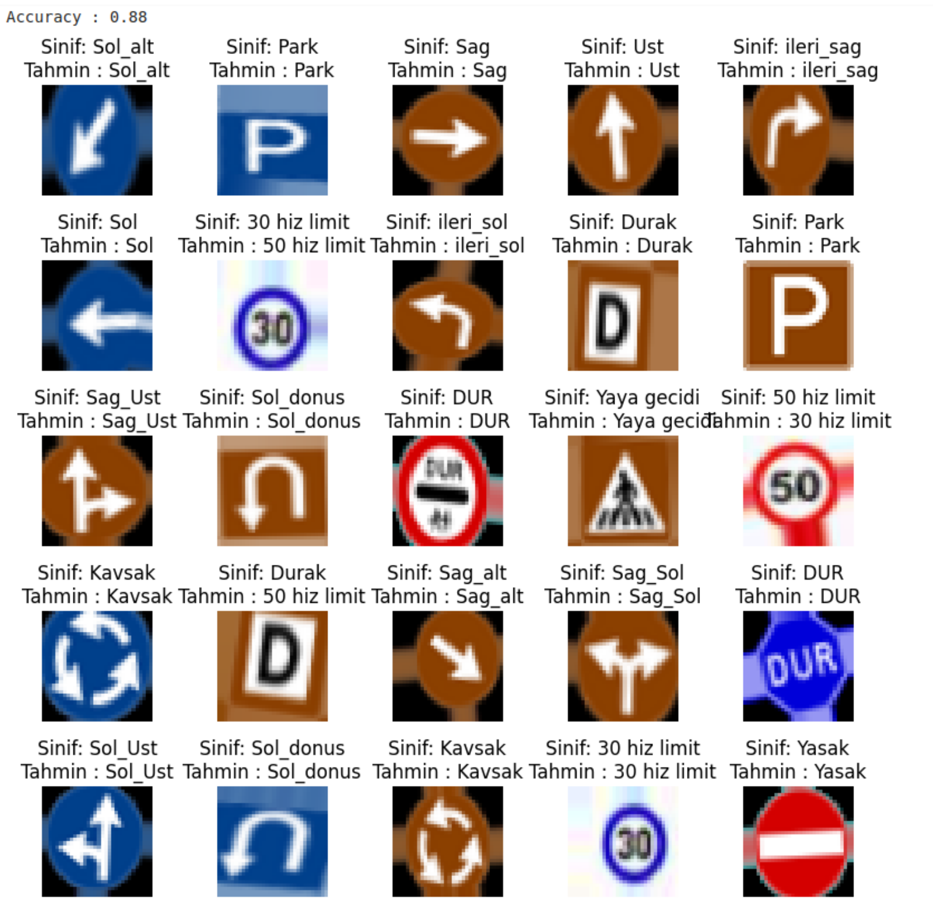


and for KNN Classifier we got this result:



```
Accuracy : 0.68
```

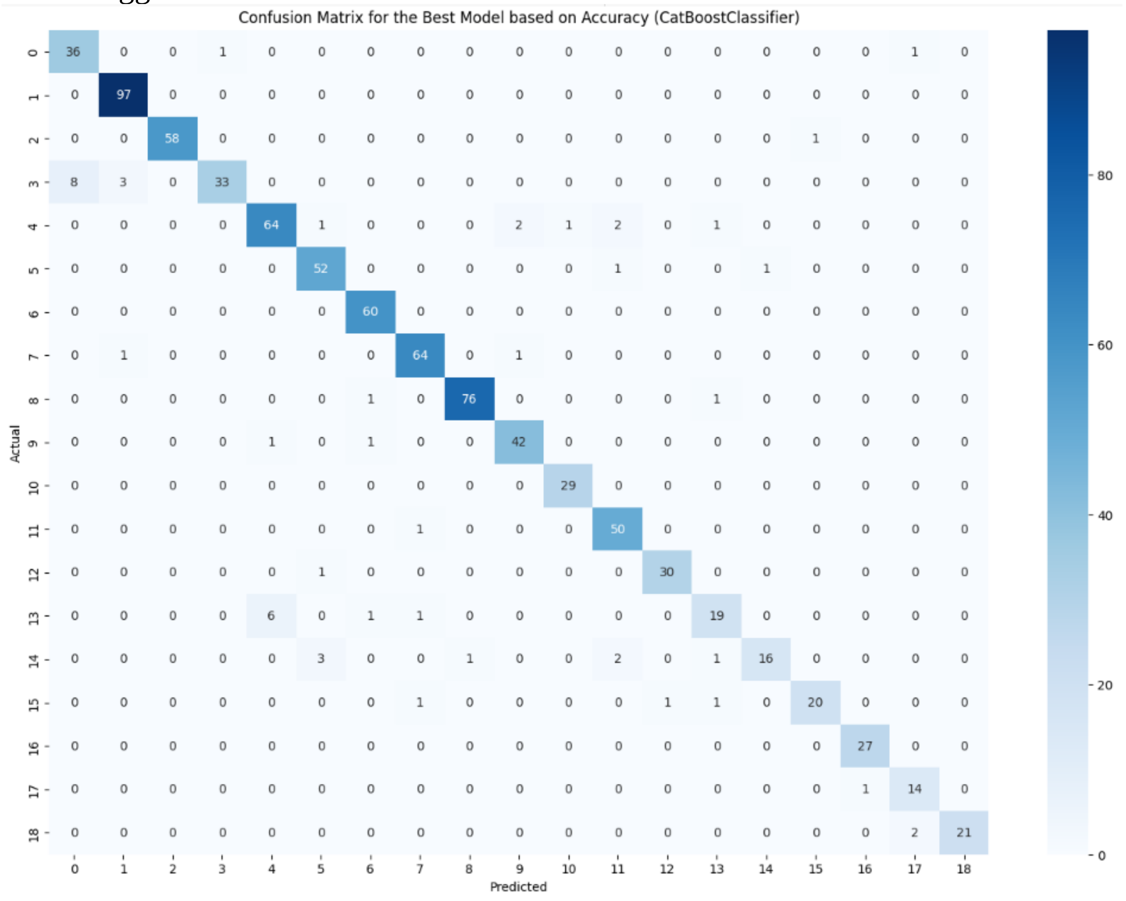
and for the last model the LGBM Classifier we got this result:



```
Accuracy : 0.88
```

Surprisingly, we discovered that skipping the feature extraction step actually made our models more efficient and accurate. We compared the results, presenting them in a clear table and graph for everyone to understand.

The next step involved diving deeper into the performance of our best model , the Cat Boost Classifier. We used a tool called the confusion matrix to see where the model excelled and where it might have struggled.

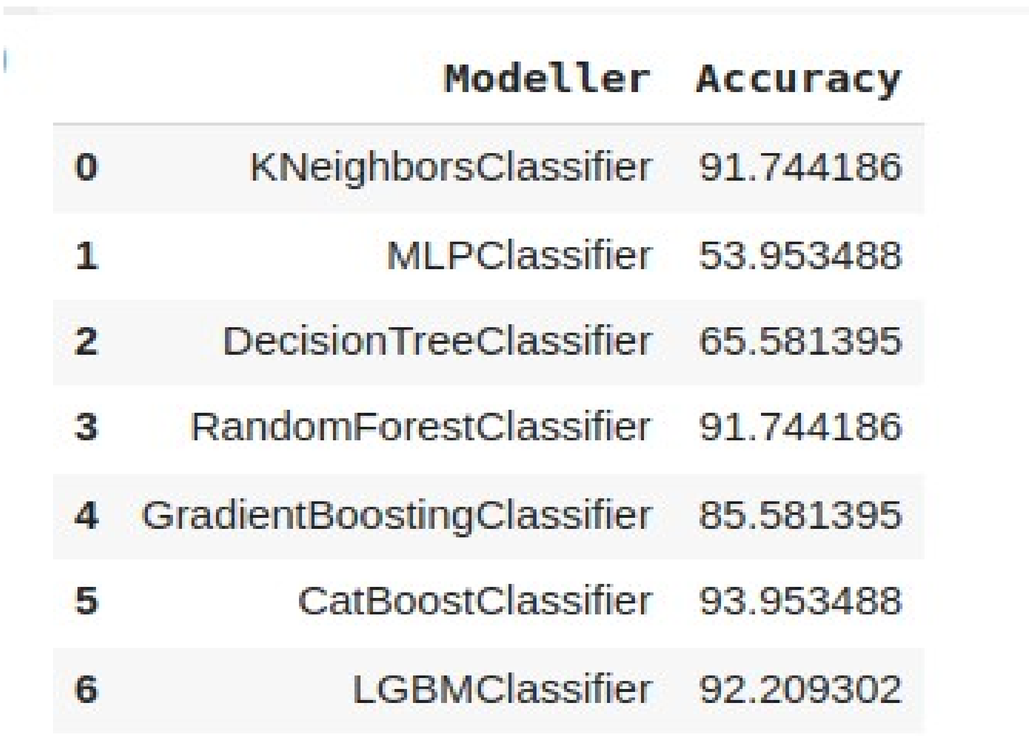


Actual \ Predicted	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	36	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	97	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	58	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
3	8	3	0	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	64	1	0	0	0	2	1	2	0	1	0	0	0	0	0
5	0	0	0	0	0	52	0	0	0	0	0	1	0	0	1	0	0	0	0
6	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0
7	0	1	0	0	0	0	0	64	0	1	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	76	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	1	0	1	0	0	42	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	29	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	1	0	0	0	50	0	0	0	0	0	0	0
12	0	0	0	0	0	1	0	0	0	0	0	0	30	0	0	0	0	0	0
13	0	0	0	0	6	0	1	1	0	0	0	0	0	19	0	0	0	0	0
14	0	0	0	0	0	3	0	0	1	0	0	2	0	1	16	0	0	0	0
15	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	20	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	14	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	21

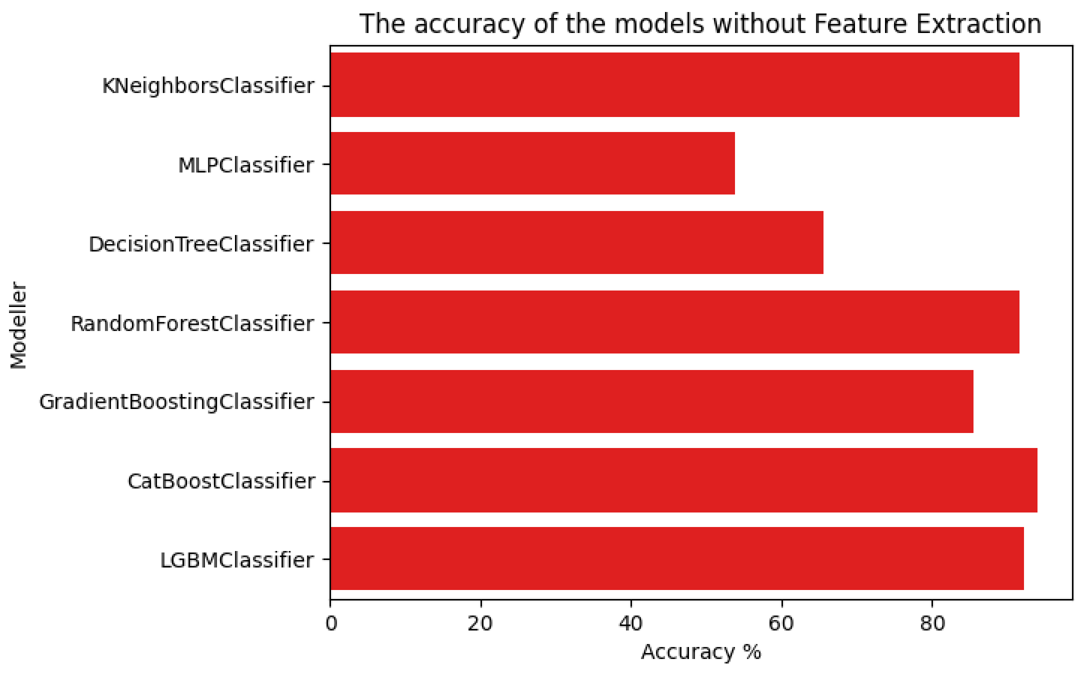
Continuing with our exploration, then this step showcased predictions made by our top best models. This allowed us to see how well they could identify and classify traffic signs.

We took these results after implementing the models without features extraction and we got also best results for classification and accuracy :

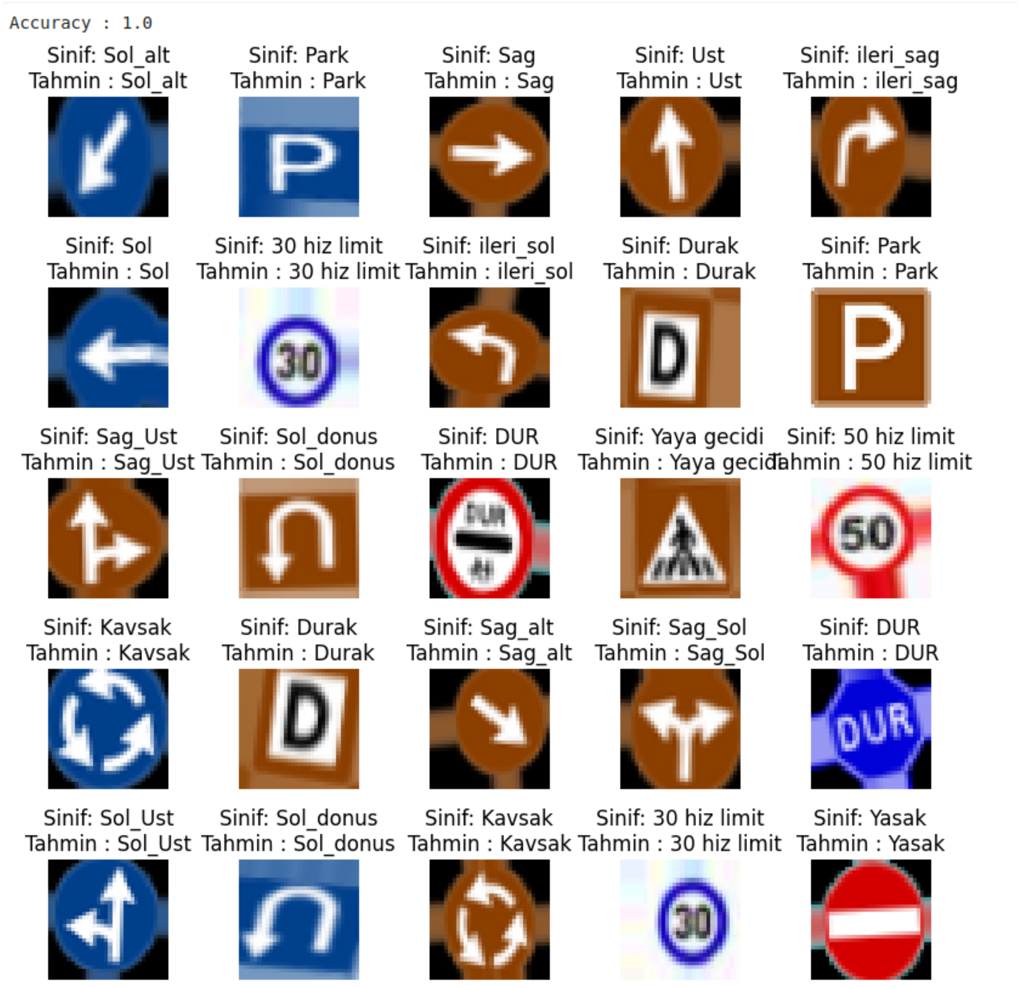
And for the model Accuracy we got these results , which obviously better than doing features extraction , these are the results :



	Modeller	Accuracy
0	KNeighborsClassifier	91.744186
1	MLPClassifier	53.953488
2	DecisionTreeClassifier	65.581395
3	RandomForestClassifier	91.744186
4	GradientBoostingClassifier	85.581395
5	CatBoostClassifier	93.953488
6	LGBMClassifier	92.209302

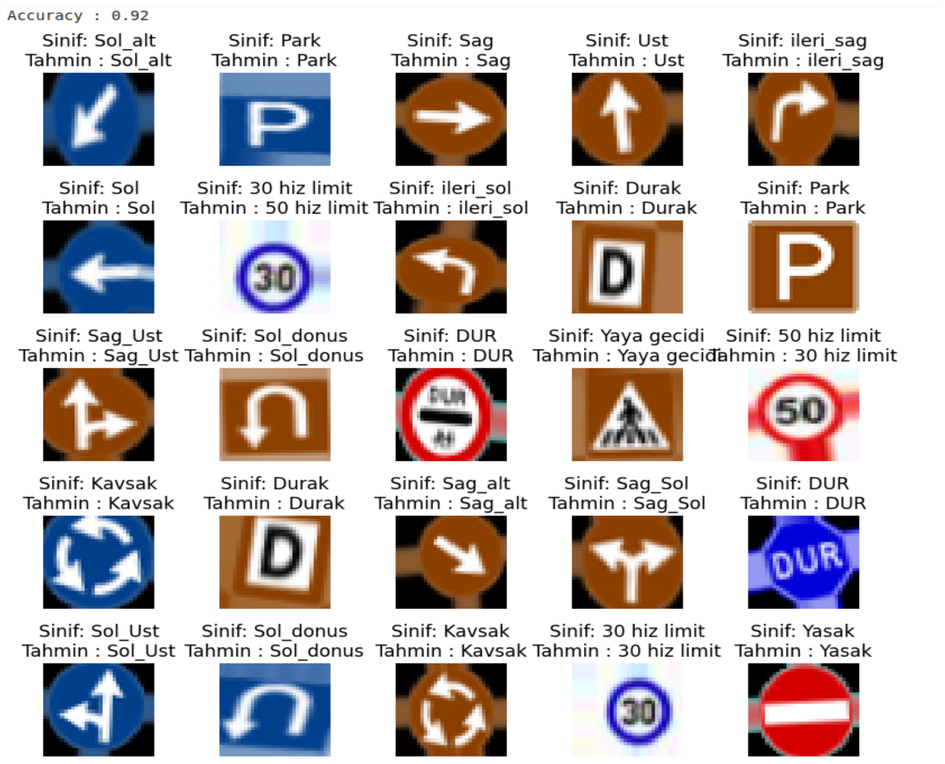


This is the results for our best three models in classification :
Cat Boost Classifier:



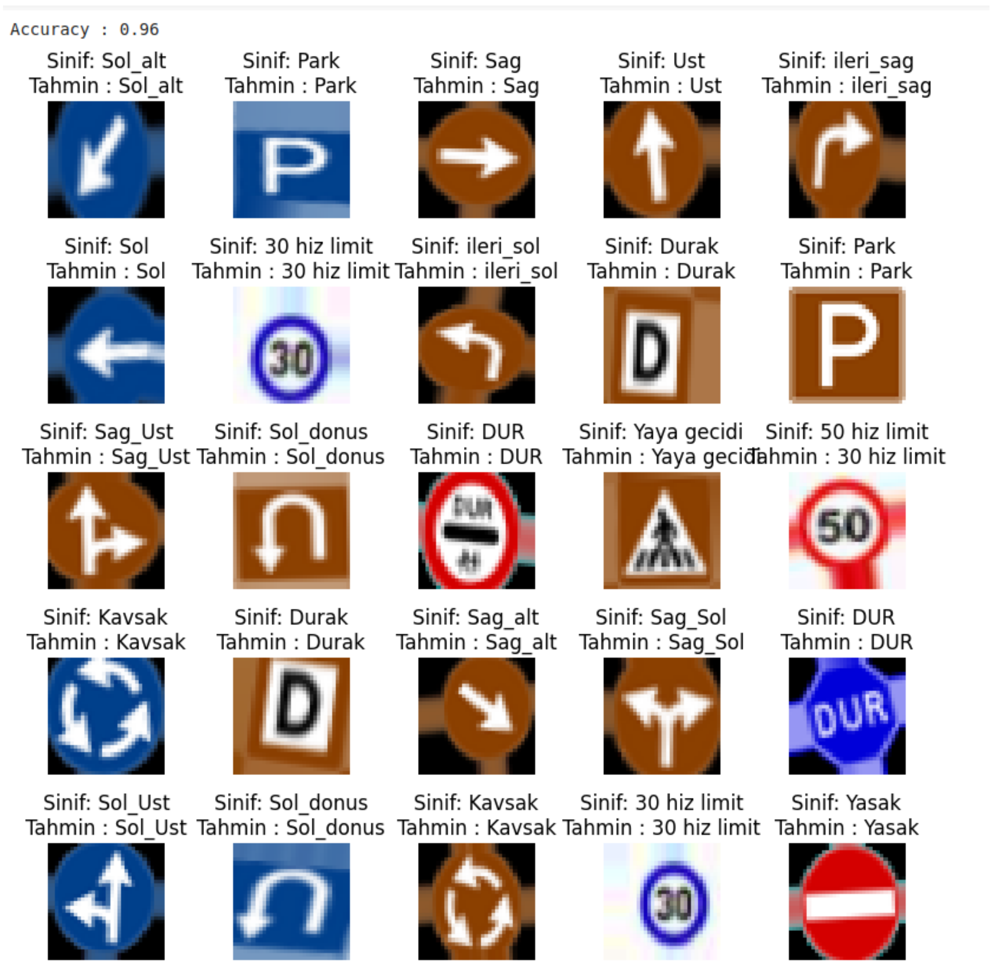
```
Accuracy : 1.0
```

Knn Classifier :



```
Accuracy : 0.92
```

for the LGBM Classifier :



```
Accuracy : 0.96
```

Recognizing the critical importance of refining our models, we fine-tuned them in the eleventh step to discover optimal settings for each algorithm, ensuring heightened accuracy. Utilizing grid search, particularly for the K Neighbors Classifier, and Cat Boost Classifier enabled a meticulous tuning process tailored to the task at hand.

In simpler terms, tuning is akin to discovering the perfect configuration for our models. This is crucial for achieving better predictions, preventing the models from being overly complex or too simplistic, allowing adaptation to diverse data characteristics, improving efficiency, addressing specific data challenges, and ensuring the models stay current with new data. In essence, tuning is a pivotal step ensuring our models are not only accurate but also precisely suited for the task at hand.

For the K Neighbors Classifier we add it with this code :

```
  knn_params = {
   "n_neighbors" : np.arange(1,15)
}
```

and for the Cat Boost Classifier we add it with this code :

```
catb_params = {
   "iterations" : [200,100],
   "learning_rate" : [0.03, 0.1],
   "depth" : [4,8]
}
```

Conclusion and References:

In conclusion, our exploration into intelligent traffic sign classification using machine learning algorithms has provided valuable insights into the capabilities of various models. From understanding the importance of feature extraction to the surprising efficiency gained by skipping this step, our journey sheds light on the complexities and nuances of the classification process.

For those interested in delving deeper into the subject, the following references provide additional context and insights:

https://link.springer.com/article/10.1007/s13369-021-05609-4/

https://www.analyticsvidhya.com/blog/2019/08/3-techniques-extract-features-from-image-data-machine-learning-python/

https://www.kaggle.com/datasets/erdicem/traffic-sign-images-from-turkey/

These resources offer comprehensive perspectives on machine learning methodologies, feature extraction techniques, and the broader landscape of intelligent transportation systems. As technology continues to advance, the fusion of machine learning with real-world applications holds the promise of creating safer and more efficient roadways.