Project 1 overdue

Bouruc Petru-Liviu

31/08/2023

Contents

1 Introduction & Project Description

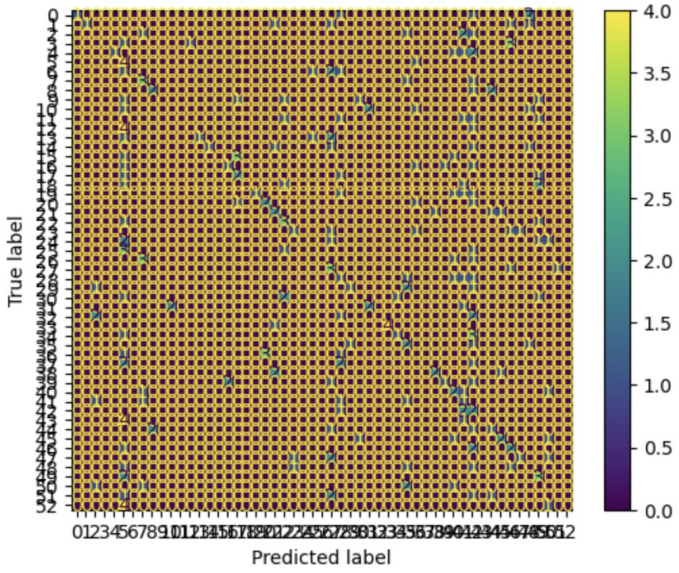
This project was about applying supervised methods on a classification or regression task. For this one, I chose to classify playing cards images. The dataset contains 7624 training images, 265 test images and 265 validation images, each partitioned into 53 types of cards. To solve this task, I tried a couple of models: Naive-Bayes, Random Forest, SVM and CNN.

For the classical machine learning algorithm, I tested with the images unchanged, using as features the original pixels, but I also tried to extract features from the HSV representation of the images. The HSV model is the most accurate representation of how humans perceive color. More than that, after converting to HSV I took the mean value of the pixels and flatten the image in order to have less computations to perform.

2 Naive-Bayes

For the first approach, I tried the simple Naive Bayes classifier, which is based on Bayes' theorem. For the implementation, I used GaussianNB, other variants being MultinomialNB and CategoricalNB, but they were more suited for datasets where the features are clearly

categorical or discrete. Using the base pixel features, the model gave an accuracy of 0.21, but when switching to the HSV features, the accuracy dropped to 0.15.



3 Random Forest

The second approach utilises the Random Forest model, an ensemble which is made of decision trees. Each decision tree anlysis a batch of the input and outputs a class regarding the classification task, and the overall output is the class that has been chosen by the majority of the trees. For this model, I used $GridSearcCV$ which allows me to make a dictionary with all the hyperparameters I want to test. It can also perform cross-validation on each run.

Hyperparameters tested:

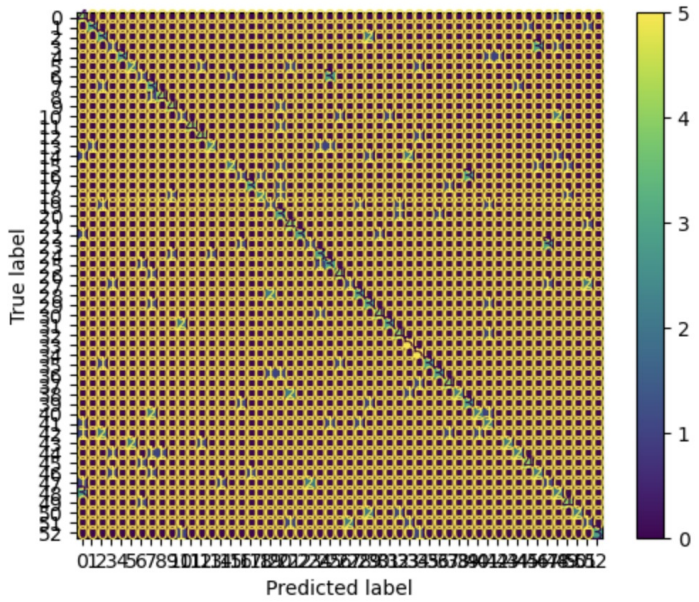
- $n_estimators$: number of trees in the forest
- $max\_features$ the number of features to consider for the best split

The results (standard):

	n = 100	n = 200
max_features=sqrt	0.53	0.51
max_features=log2	0.44	0.47

The results (HSV features):

	n = 100	n = 200
max_features=sqrt	0.41	0.45
max_features=log2	0.38	0.40



4 SVM (Support Vector Machines)

As a third approach, I tried a classifier based on Support Vector Machines (SVM). This is a binary classifier which uses a delimitation between two margins to maximize the data points that should be correctly classified. To use it for multiclass classification, it uses a technique such as one-vs-one method, the final result being the pair of classes that when compared each, had the best result.

For the training itself, I used the *SVC* module from sklearn alongside *GridSearcCV*. Hyperparameters tested:

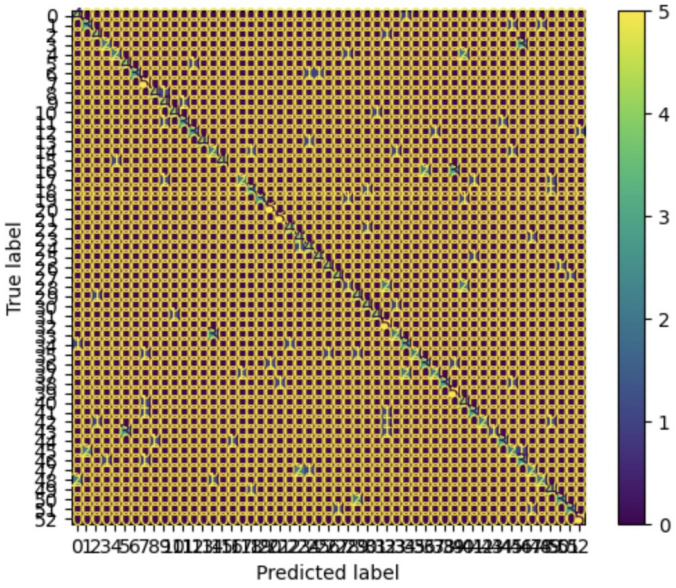
- Regularization parameter C: tells the model how much error it can allow when classifying the data; a large C reduces the learned margins (can lead to overfitting) / a small C is more permissive (can lead to underfitting)
- *kernel* parameter has the purpose to project the data to linearly separable feature spaces and represents the type of kernel the model used for this

The results (standard):

	n = 100	n = 200
max_features=sqrt	0.53	0.51
max_features=log2	0.44	0.47

The results (HSV features):

	n = 100	n = 200
max_features=sqrt	0.41	0.45
max_features=log2	0.38	0.40



5 Convolutional Neural Network

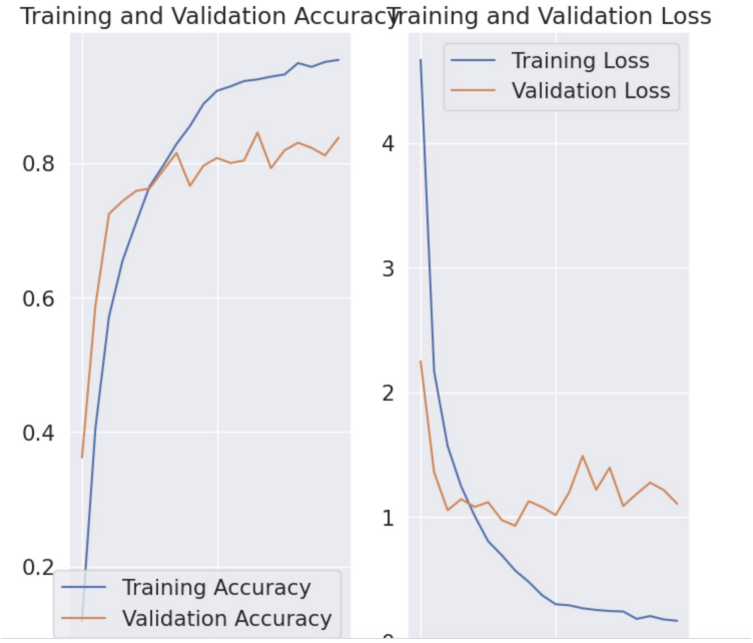
In the end, I tried to approach this problem with a deeplearning method for comparison, so I used a convolutional neural network because these are suited for images. It is composed from multiple layers, each of them having a different role.

One of the core layers for convolutional neural networks is $Conv2D$ layer which uses a filter and a kernel size. With those, runs on the image, multiplies the pixels of the image with the filter and reduces the size of the window data based on the kernel size. The learnable parameters are on the filter. As for the activation function, I chose ReLU (y = max(x, 0)), an advantage being that it is really easy to compute and fast.

Another layer used is $MaxPooling2D$ which takes the maximul value from a given window, reducing the complexity and also keeping the relevant information. Alongside this, in the architecture I also used Dropout, which have the role of randomly dropping previously learned features, reducing the risk of overfitting, and Flatten with Dense layers for liniarization, the last layer having softmax activation function which transforms the result into an array of probabilities for each class.

Regarding the training, for this approach, I used the build-in function from keras to read the dataset: $image\_dataset\_from\_directory$. It is suited for datasets structured like this one.

In the end, I tried multiple combinations of these layers in order to find the configuration with the best results. Changes in the learning rate didn't improved the accuracy, but had an impact on how fast my model could learn, and also too little values could lean to overfitting. The optimizer used was *Adam*, which simulates the Gradient Descent algorithm. I managed to get a mean accuracy of 0.83 on the validation and test set, with the following loss and accuracy evolution:



6 Comparison with other methods

Looking at methods other people used, I observed on Kaggle that many of them approached this dataset using a transfer-learning method. This involves taking a trained architecture and adapting it to the dataset we want to train (via re-training it, adding new layers or both). The favourite architecture was VGG16 or EfficientNetB6, giving results such as 92, 96 or 96 % accuracy.

7 Conclusions

From all the tests I made and from the comparison with other methods, we can see that the deeplearning approaches have the best result, especially those which employs transfer learning.