Music Instrument Classification using Machine Learning Techniques

Master's degree in Computer Engineering for Robotics and Smart Industry

Denis Tognolo

AY 2022/2023

Contents

1 Introduction

This work goals to develop a classificator for music instruments using different machine learning techniques and compare them. In particular I tried four different techniques for feature extraction and two classification models, all of them tuned properly in their parameters, in order to find the most performing solution.

Audio analysis and classification have become very popular researches in the digital world, for both entertainment and business purposes. The fundamental characters of the tones produced are pitch, loudness, duration and timbre. One of the basic functions of a musical instrument is to produce tones of the desired pitch [1]. For music analysis the most popular task are genre classification [2], mood classification [3], instrument identification [4] and music tagging [5].

But also audio signal classification in general is a very a popular topic, in particular for speech recognition [6] or filtering [7] , but can also be used to identify and prevent failure in industrial environments [8] .

The state of the art for this kind of application consist of course in deep learning techniques, but for the purpose of this course and project I don't consider them at all, reaching anyway really good performances.

2 Implementation and Development

This work is implemented in python [9] using scikit-learn [10] for the machine learning utilities, librosa [11] for audio files management and matplotlib [12] for the visualization. The main idea behind the software architecture is to have an *instrument_classificator.py* that collect all the functions and then some other scripts (*main_KNN.py* and *main_SVM.py*) to build a particular model using them. Then I also setup some bash files (*run_KNN.sh* and *run_SVM.sh*) to run them several times changing some parameters. At the end of each test all the metrics (see section 6), are automatically saved in a text file including the considered parameters. Also the figure representing the confusion matrix (see section 6.1) is saved at the end of each test. Whit this setup, I crafted and run a total of more than 60 test with different configurations.

3 Dataset and Pre-Proccesing

In order to train the models I'll present you in the next sections, a large labelled dataset was needed. For this work I used the free philarmonia orchestra sound samples [13] that provides over 13000 samples from 15 different instruments.

Actually, all the test I present you in this report, are done in a subset of it that consist in 12 classes with 300 samples for each of them. This choice goals to consider a more balanced dataset, and also to solve some computation issue due to too much large matrix during some of the training.

All the audio files in the philarmonia orchestra dataset are loaded and labeled according to their containing folder. Then they are set to a fixed duration and

sample rate so that all the samples have same dimensionality. After that I trimmed leading/trailing silence, by removing initial and final portions of the sample which loudness is below a certain threshold (in this case 20 dB). Then all the samples are normalized by dividing each frame by each sample maximumn value. Finally the dataset is randomly split into training and test set according to a given rateo.

4 Feature Extraction

In this work I compared some different feature extraction techniques that I'll explain you in details in the following subsections.

4.1 Original Signal

A raw audio signal consist in a measure of the loudness for each time sample, usually expressed in a log scale (dB).

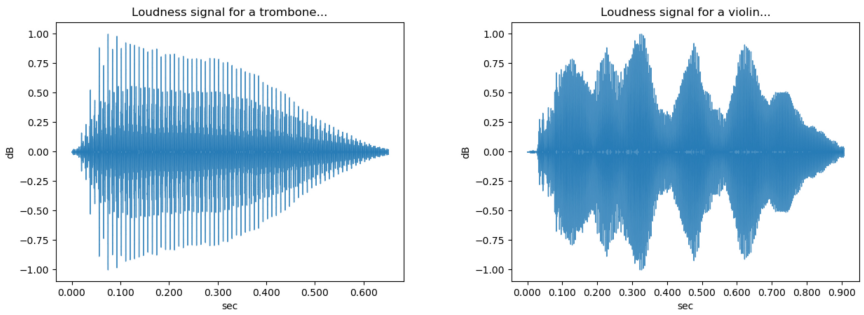

Figure 1: Example of the original signals for a trombone and a violin

For a instrument recognition task, this feature is not significant at all because it represents more the way an instrument is played (legato, pizzicato, tremolo..), than which one is playing. In fact, as I'll present you in the results section [7] the confusion matricies obtained considering no feature extraction result very non diagonal.

4.2 Spectogram

To properly menage an audio signal in the frequency domain a simple Fourier transformation applied to the whole signal is not sufficient to extract relevant information. Instead its better to divide the signal into smaller frames (windows) and so apply the Fourier transformation to each of them. This operation is called Short-Time Fourier Transform or STFT. For better performances, is

recommended to consider overlapping frames filtered by a sine wave, to somehow consider correlation between near frames. The output of this operation is a matrix in which rows represent a frequency and columns a particular time frame. Each cells value represents how much this frequency is present in that particular time frame.

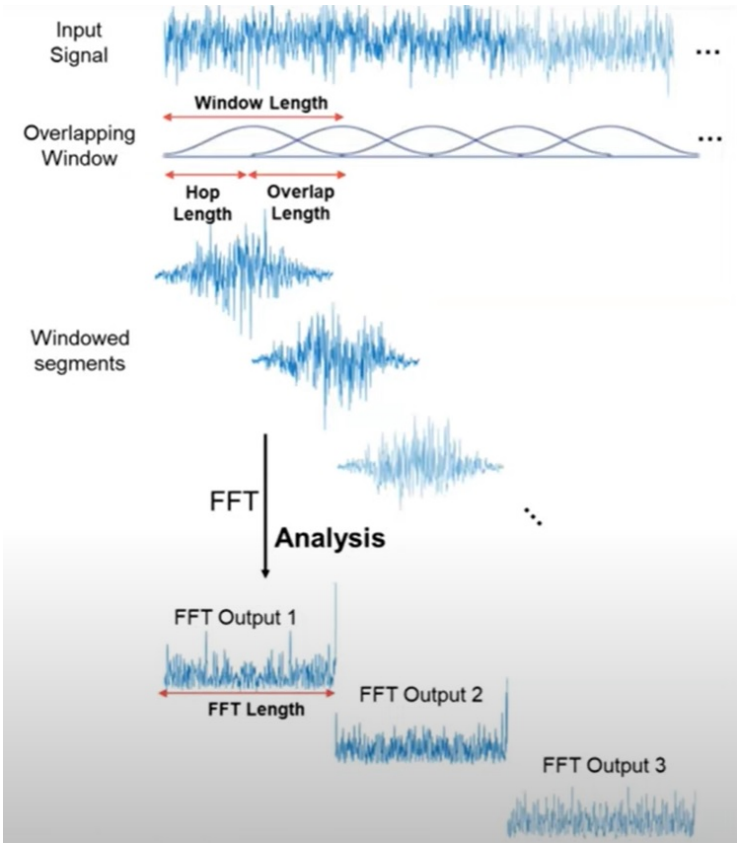

Figure 2: Intuition of Spectograms

So that the parameters to define to properly apply this transformation are the size of the frames and the overlapping time. This are the results by performing STFT with different values of this parameters.

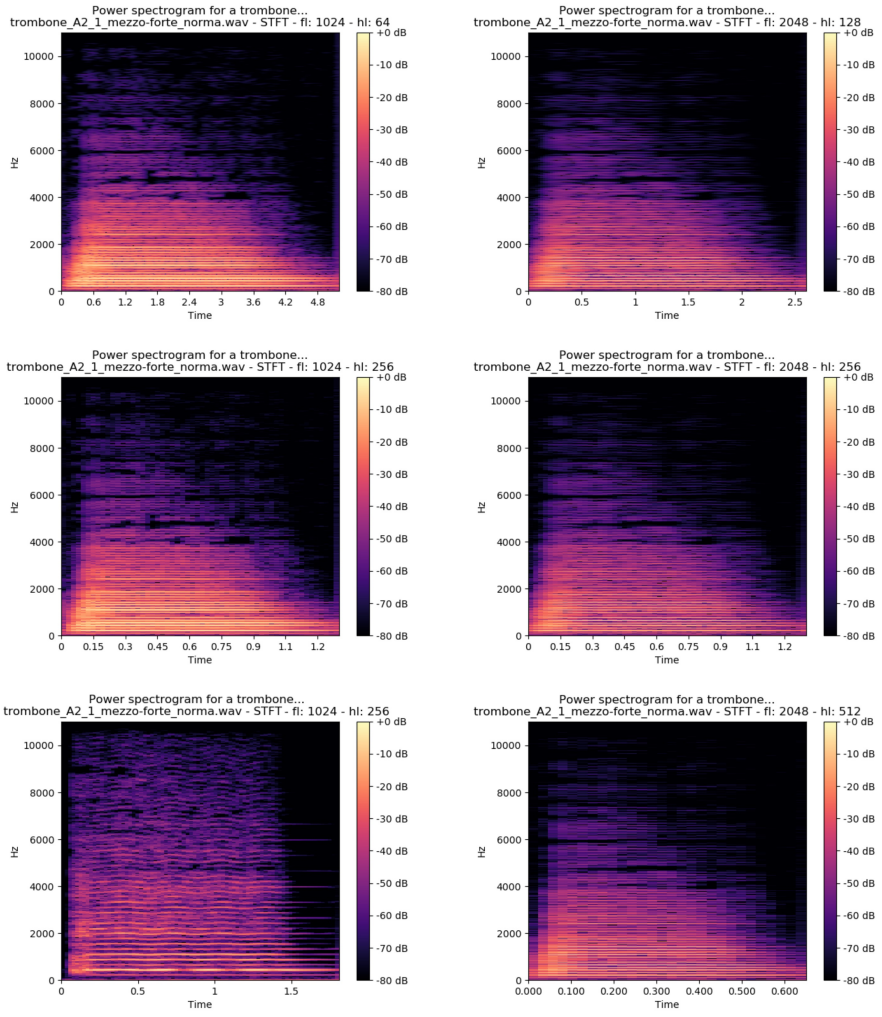

Figure 3: Example of STFT for a trombone with different frame length and hop length values

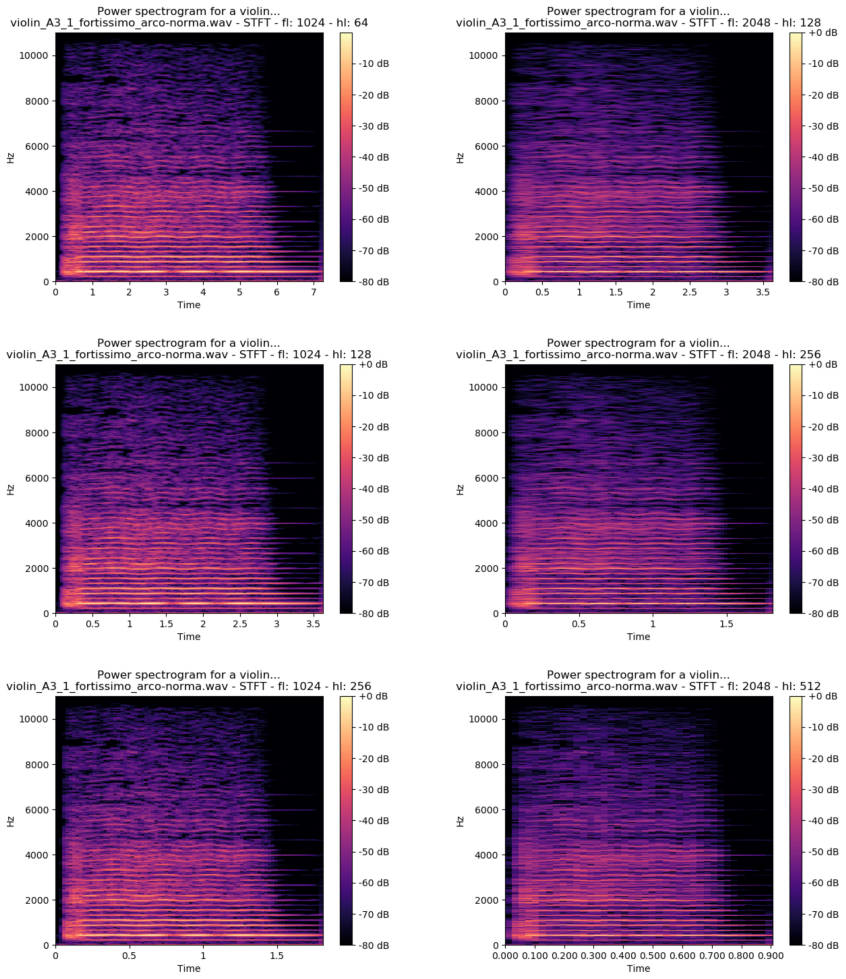

Figure 4: Example of STFT for a violin with different frame length and hop length values

4.3 Spectral Centroid

Spectral centroid indicates where the "center of mass" for a sound is located by calculating the weighted mean of the frequencies present in the sound. In particular for each frames it computes the weighted mean of the frequencies with their magnitudes as the weights.

$$SC_i = \frac{\sum_{k=1}^{N} F(k) f(k)}{\sum_{k=1}^{N} f(k)}$$

where i indicates the time frame, N the number of bins, $F(k)$ the center frequency of bin k, $f(k)$ the spectral value of that bin, and so SC_i the index of which bin is the "gravity center" for that frame.

The parameters to define to properly apply this transformation are the size of the frames, the overlapping time and the number of bins. This are the results by performing Spectral Centroid with different values of this parameters.

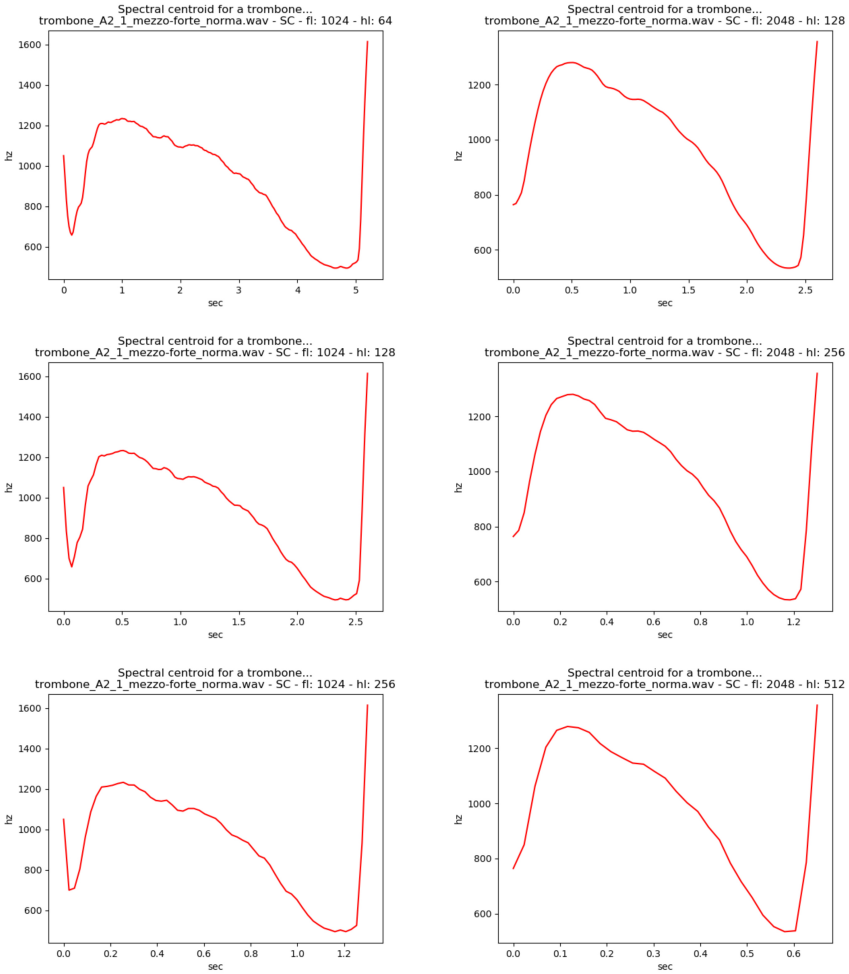

Figure 5: Example of SC for a trombone with different frame length, hop length values.

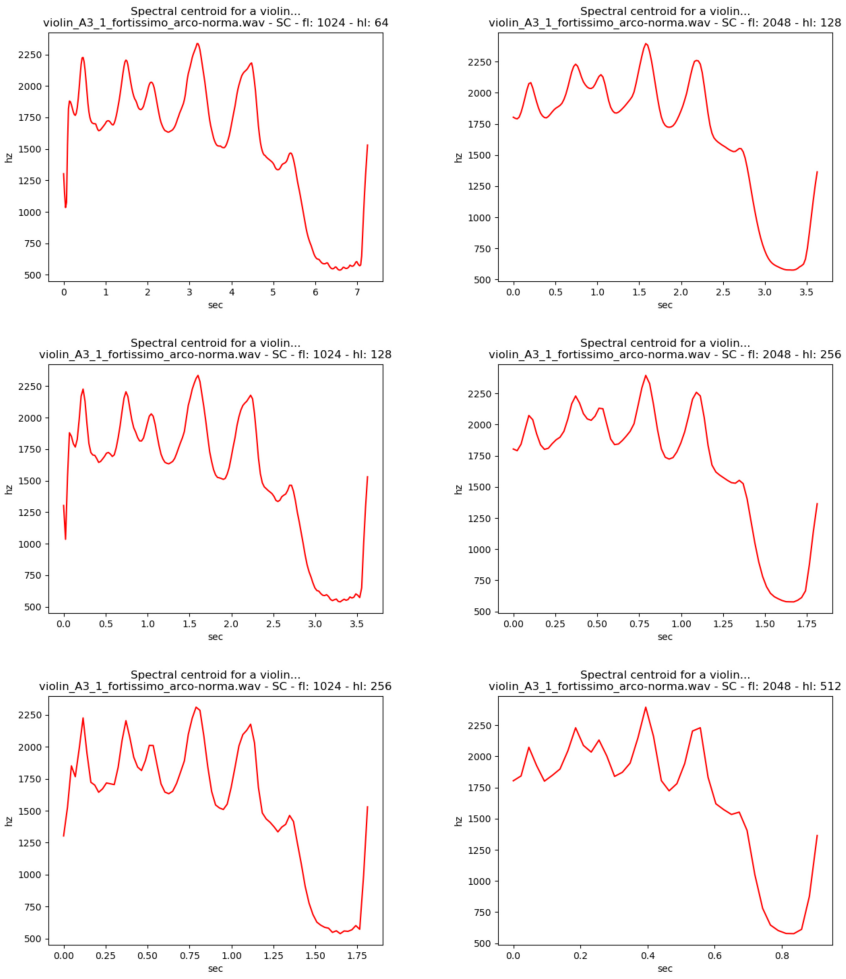

Figure 6: Example of SC for a violin with different frame length, hop length values.

4.4 Mel Spectogram

The mel scale is a perceptual scale of pitches judged by listeners to be equal in distance from one another. To convert a signal from the frequencies into the mel scale you can simply apply the following equation:

$$m = 2595 \cdot log_{10}(1 + \frac{f}{1000})$$

After this scaling the signal is usually divided into some bands. This operation can be done by filtering the frequency signal by some triangular shape that

represent a certain band. In order to consider some correlation between near bands, each of them starts at the pick of the previews one and ends at the pick of the next one.

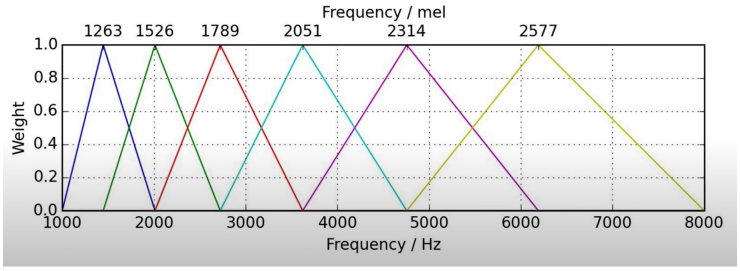

Figure 7: Example of a Mel filter for 6 bands

The parameters to define to properly apply this transformation are the size of the frames, the overlapping time and the number of mel-bands. This are the results by performing MSTFT with different values of this parameters.

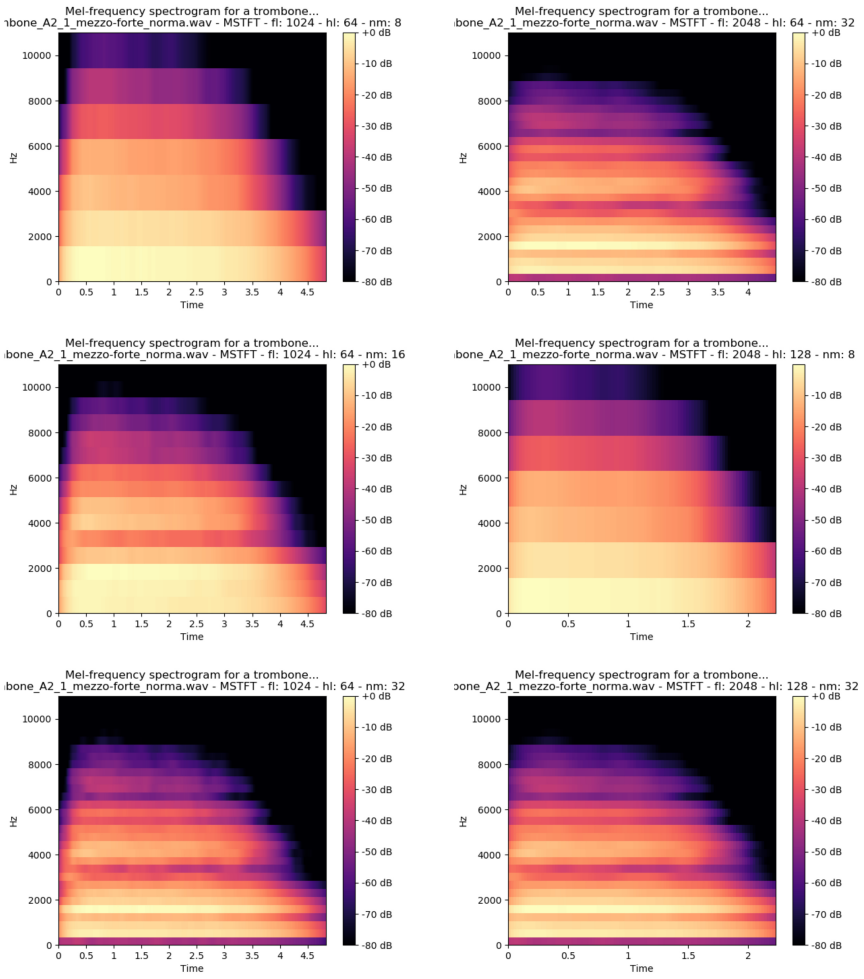

Figure 8: Example of MSTFT for a trombone with different frame length, hop length values and the number of mel-bands

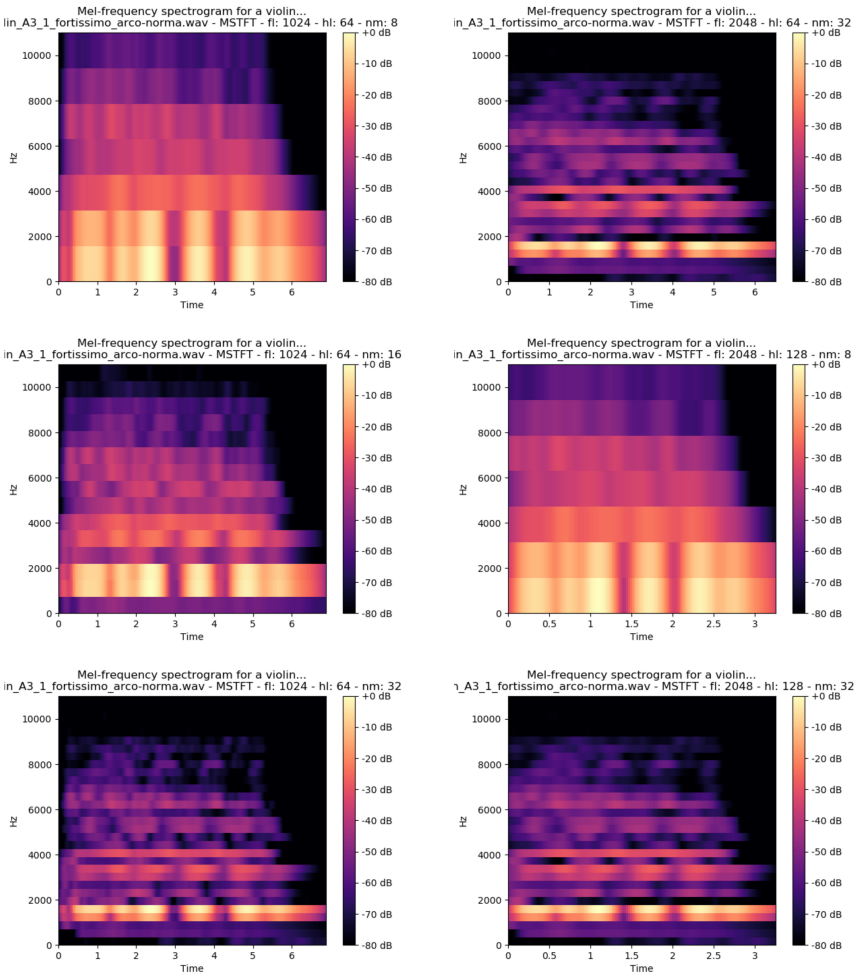

Figure 9: Example of MSTFT for a violin with different frame length, hop length values and the number of mel-bands

4.5 Mel-Frequency Cepstral Coefficients

MFCC is a tool used to construct a feature vector starting from an audio file, widely used in speech modeling. This method extracts for all frames, a small set of features (usually about 10–20) that concisely describe the overall shape of a spectral envelope (curve in the frequency-amplitude plane). MFCCs are commonly derived as follows:

1. Compute the Mel-Spectogram of a signal.

2. Take the logs of the powers at each of the mel frequencies.

3. Take the discrete cosine transform of the list of mel log powers.

The cosine transform is preferred over the classic one, because is simpler to compute, only produce real-valued coefficients and also reduce the dimensionality of the feature. In fact the dimension of the MFCC will be determined by the order of the Taylor approximation of the cosine considered during this transformation, or in other worlds, the number of cosine we try to fit into the signal. The results are the amplitudes of the resulting spectrum or better the cepstrum of the signal.

The parameters to define to properly apply this transformation are the size of the frames, the overlapping time and the number of mel-bands. This are the results by performing MSTFT with different values of this parameters.

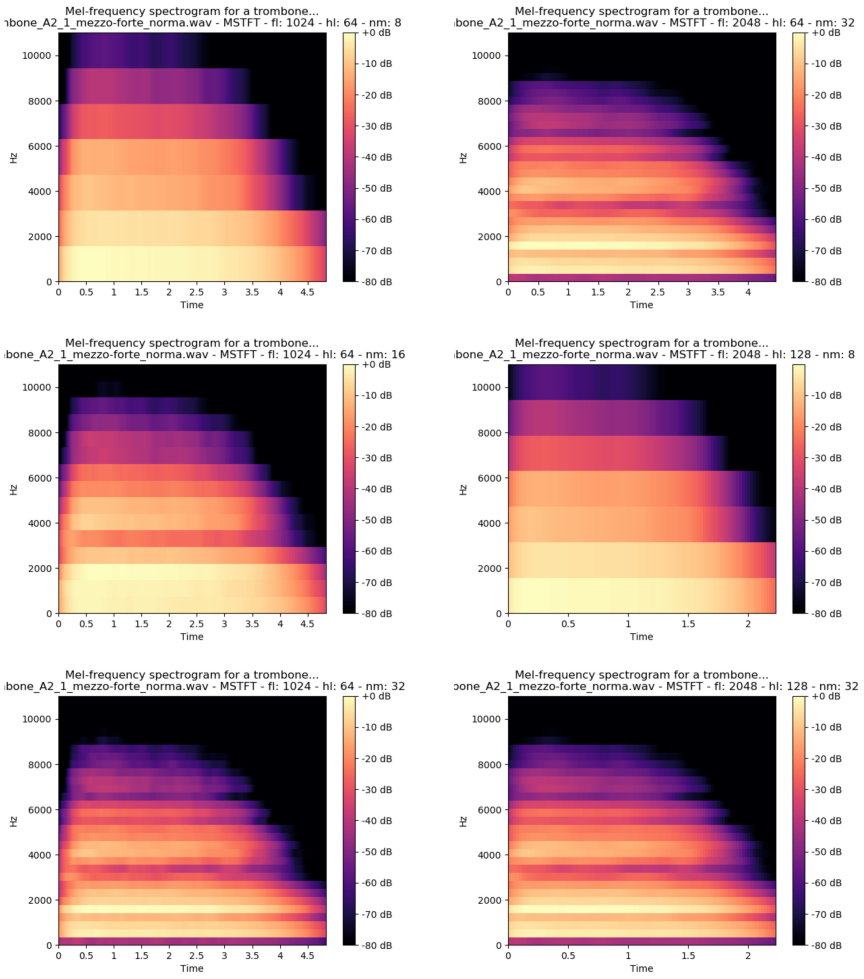

Figure 10: Example of MSTFT for a trombone with different frame length, hop length values and the number of mel-bands

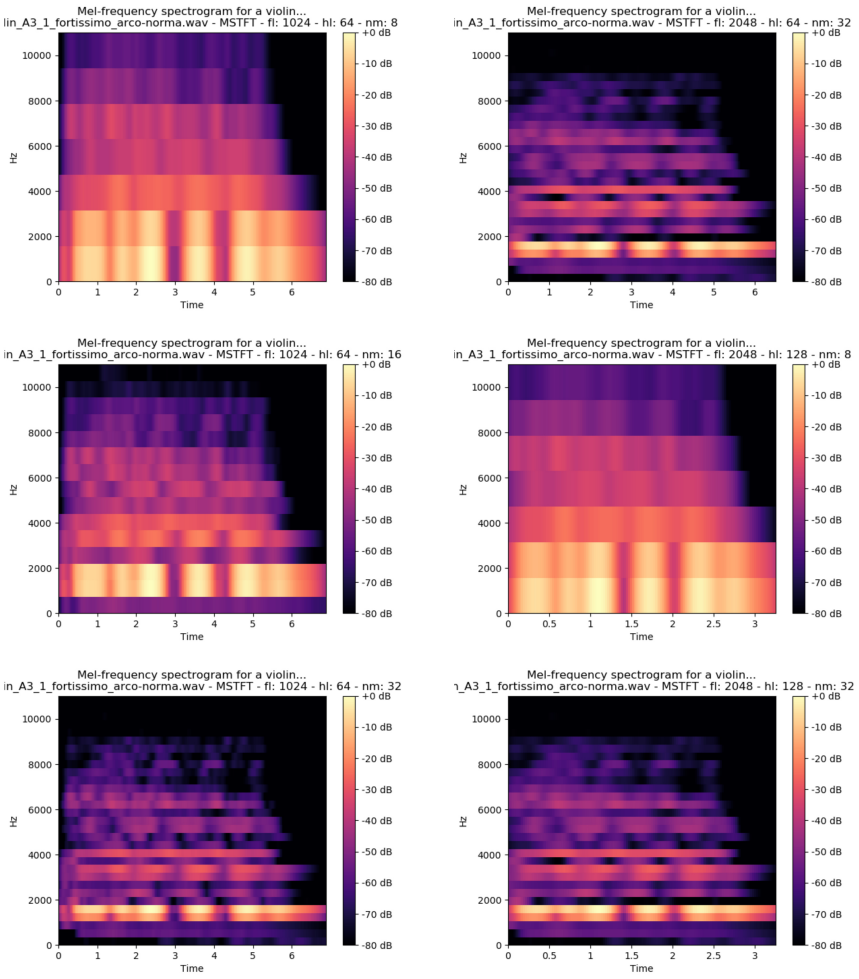

Figure 11: Example of MSTFT for a violin with different frame length, hop length values and the number of mel-bands

5 Classification

In this work I compared two different classification techniques that I'll explain you in the following subsections.

5.1 K-Nearest Neighbors

The k-nearest neighbors algorithm is a supervised learning technique, which uses proximity to make classifications working off the assumption that similar points can be found near one another. For classification problems, a class label is assigned to the one that is most frequently represented around a given data point.

The k value in the k-NN algorithm defines how many neighbors will be checked to determine the classification of a specific point. For example, if k=1, the instance will be assigned to the same class as its single nearest neighbor. Defining k properly can avoid overfitting or underfitting, in fact lower values of k can have high variance, but low bias, and larger values of k may lead to high bias and lower variance. The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k.

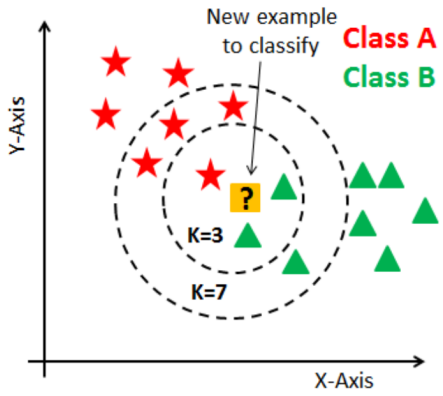

Figure 12: Intuition of KNN

5.2 Support Vector Machine

Support vector machines are supervised learning models used to analyze data for classification and regression analysis. The main idea is to divide data into two categories, by placing an hyperplane between them, in particular trying to maximize the distance from the nearest data from both groups (better known as support vectors). Note that this technique works only for a one vs one classification task. So in order to perform a multi class problem, n models are individually trained and during the evaluation, after a sample is evaluated for all of them, the class with the higher result is considered as the belonging one.

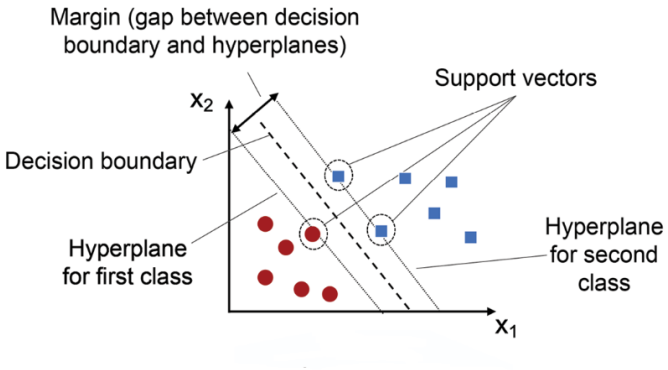

Figure 13: Intuition of SVM

The hyperplane that forms the margin can be of different types (linear, polynomial or even more), depending on the data distribution. In practices, in order to avoid to search for a curve that could potentially requires an infinite number of parameters, we achieve this goal by performing a transformation to all the data before the surface fitting. This operation is known as the kernel trick.

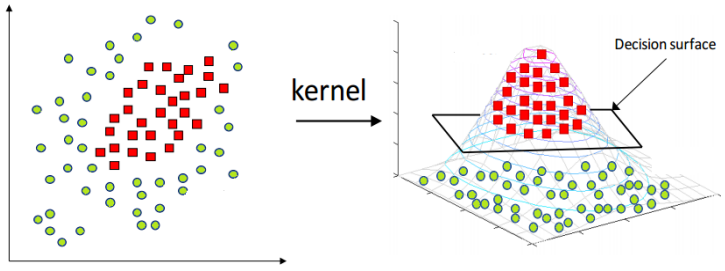

Figure 14: Intuition of the Kernel Trick

6 Evaluation Metrics

In order to evaluate how good a considered model or solution performs, I compute for each of them a confusion matrix, comparing the real labels (ground truth) with the predicted ones.

6.1 Confusion Matrix

A confusion matrix is a table that is used to define the performance of a classification algorithm, visualizing and summarizing its performance. For a N classes classificator, this matrix dimension results $N \times N$ where each column represent a predicted class and so each raw a real one. So the cell at raw i and column j represents how many times our model classified a sample that belong at class i as class j. Of course the sum over the major diagonal represent all the correct predictions.

From a confusion matrix, that provide a visual interpretation of the performance, we can compute some numeric metrics that I'll present you in the next subsections.

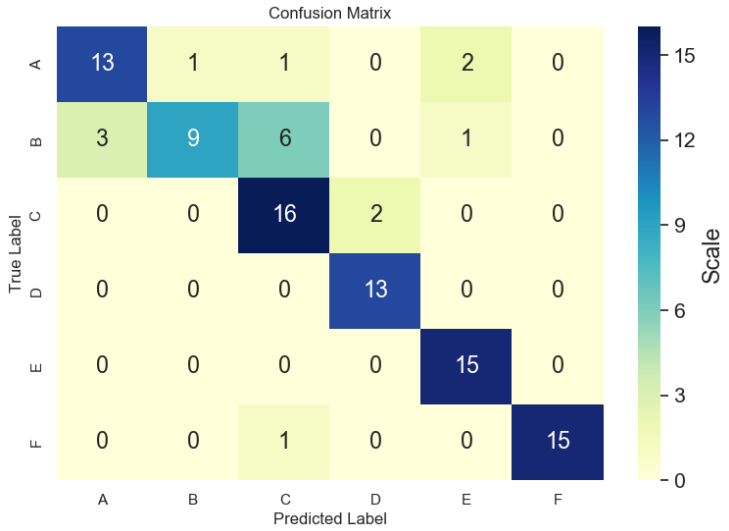

Figure 15: Example of a confusion matrix for a N=6 classes classification

6.2 Accuracy

Accuracy, is the simpler metric, and consider the rateo between the number of correct prediction over the total number of them.

$$Accuracy = \frac{TP}{TP+TN+FP+FN} = \frac{RightPredictions}{TotalPredicitons}$$

6.3 Precision

For a class i, the precision represents the rateo between the number of correct predictions over the total number of predictions for that class, considering also false positive.

$$Precision_i = \frac{TP}{TP+FP} = \frac{RightPredictions_{class_i}}{TotalPredictions_{class_i}}$$

The precision for the whole model is defined as the mean over all the classes.

6.4 Recall

For a class i, the recall represents the rateo between the actual number of correct prediction over the total number of correct prediction that the ideal model should produce.

$$Recall_i = \frac{TP}{TP+FN} = \frac{RightPredictions_{class_i}}{RealMembers_{class_i}}$$

The recall for the whole model is defined as the mean over all the classes.

6.5 F-Score

F-score is a tool to embed both precision and recall information in a single value and is computed by the harmonic mean of them.

$$F_{score} = 2 \times \frac{Recall \times Precision}{Recall + Precision}$$

Note that for a F-score near to one, the system perform almost perfectly, and really bad for values near to zero.

7 Results

A comparison of the evaluation metrics for both KNN and SVM, considering same feature extraction method, highlights the difference between the two solutions. In particular, as I'll better explain you in the next subsections, KNN results to work properly with any of the considered feature, even with the largest ones without making the execution to become really slow. That's not the same for SVM, that due to the fact that it consider more than one model (one for each class), it really suffer on time performances while managing large feature. On the other hand the results for the slow runs are excellent, making the tuning of its parameter, in order to find a trade off between performance and speed, really appreciable.

7.1 KNN Results

In order to find the best setup to perform the instrument classification task using KNN, first at all I fixed the number of neighbours ($nn = 5$) and run some test for each feature. Then, after I found the best one, I try to optimize the KNN parameter, in particular the number of neighbour to consider.

7.1.1 KNN + Signal

As expected and discussed in section [4.1], the original signal is not relevant for this task, and in fact the confusion matrix result not diagonal at all.

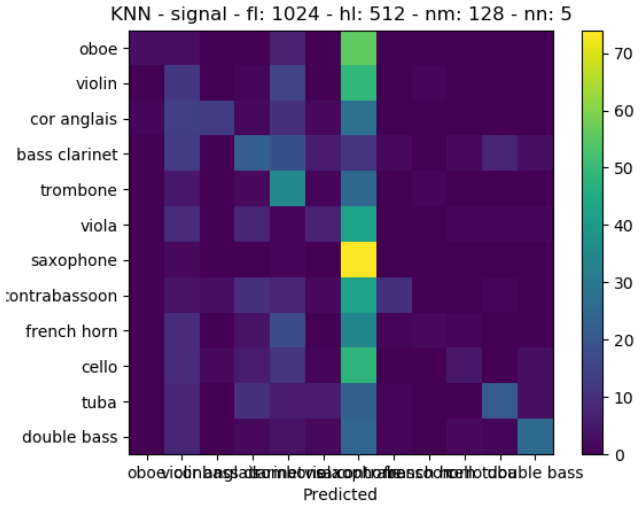

Figure 16: Confusion matrix for KNN + original signal

accuracy: 0.88 | precision: 0.47 | recall: 0.26 | fscore: 0.23

7.1.2 KNN + STFT

As I discussed in section [4.2], the tuning parameters for this feature are the frame and hop lengths. Here the results for some different values of them:

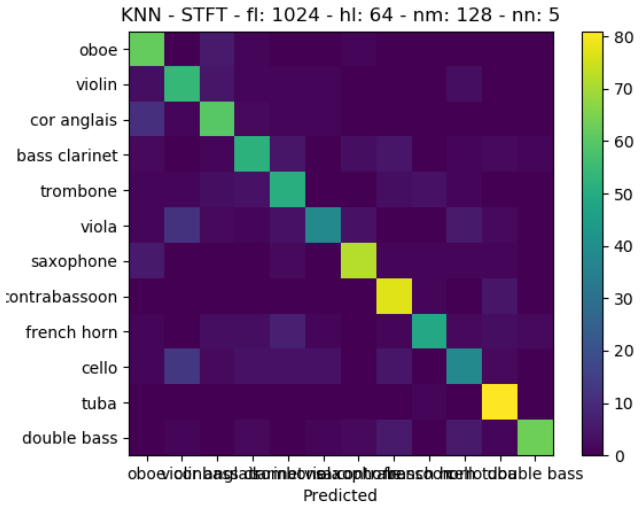

Figure 17: Confusion matrix for KNN + STFT (fl = 1024, hl = 64)

accuracy: 0.96 | precision: 0.78 | recall: 0.77 | fscore: 0.77

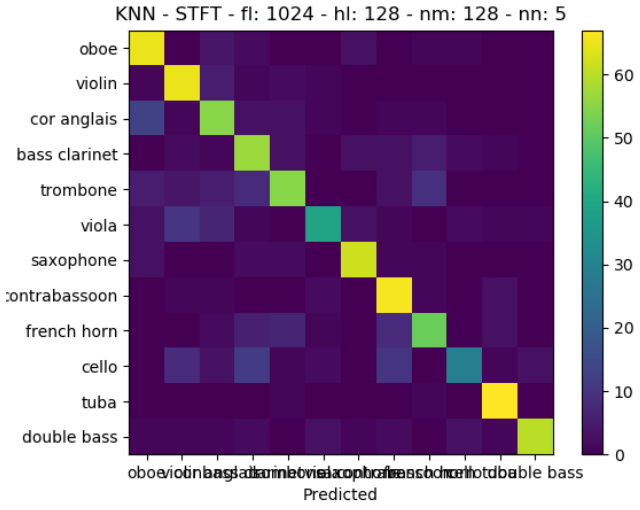

Figure 18: Confusion matrix for KNN + STFT (fl = 1024, hl = 128)

accuracy: 0.96 | precision: 0.76 | recall: 0.75 | fscore: 0.74

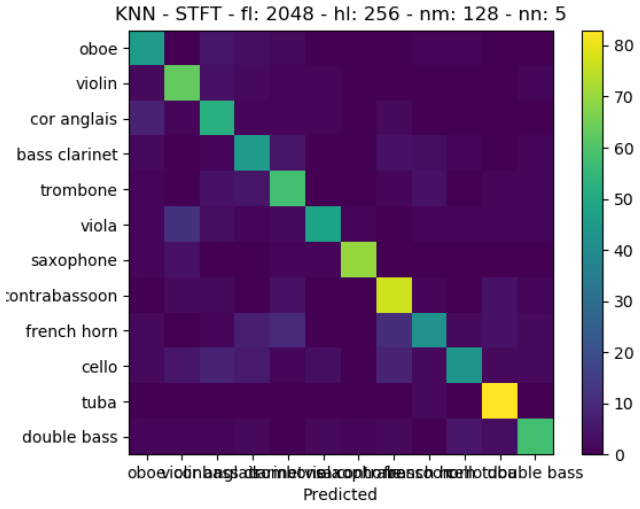

Figure 19: Confusion matrix for KNN + STFT (fl = 2048, hl = 256)

accuracy: 0.96 | precision: 0.77 | recall: 0.76 | fscore: 0.76

As we can note there's not a big difference between the different solutions, that overall can be considered very good.

7.1.3 KNN + SC

As I discussed in section [4.3], the tuning parameters for this feature are the frame and hop lengths and also the number of bins. Here the results for some different values of them:

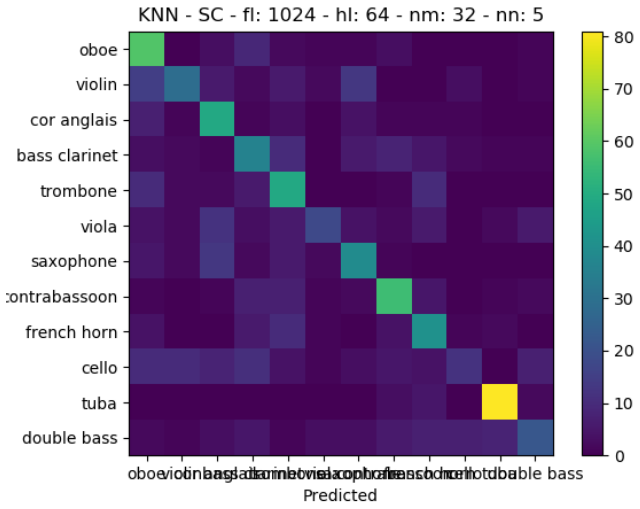

Figure 20: Confusion matrix for KNN + SC (fl = 1024, hl: 64)

accuracy: 0.92 | precision: 0.55 | recall: 0.53 | fscore: 0.52

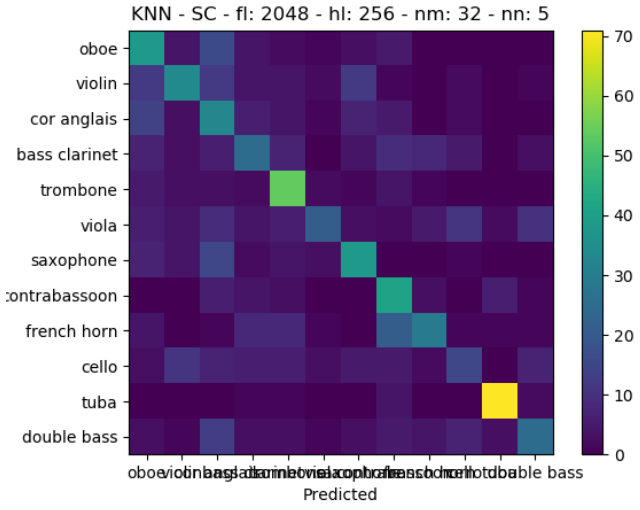

Figure 21: Confusion matrix for KNN + SC (fl = 2048, hl: 64)

accuracy: 0.91 | precision: 0.48 | recall: 0.47 | fscore: 0.46

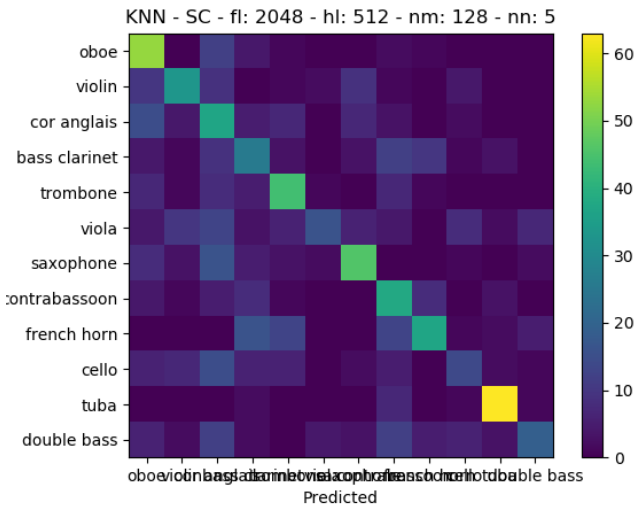

Figure 22: Confusion matrix for KNN + SC (fl = 2048, hl: 512)

accuracy: 0.91 | precision: 0.5 | recall: 0.47 | fscore: 0.46

As we can note there's not a big difference between the presented solutions, that overall are not good as the STFT ones. But it's important to note that this time the features are so much shorter than the STFT ones, making this solution more appreciable in terms of complexity. Anyway with KNN this difference in the execution time is not enough to consider this solution instead of STFT.

7.1.4 KNN + MSTFT

As I discussed in section [4.4], the tuning parameters for this feature are the frame and hop lengths and also the number of mel-bands. Here the results for some different values of them:

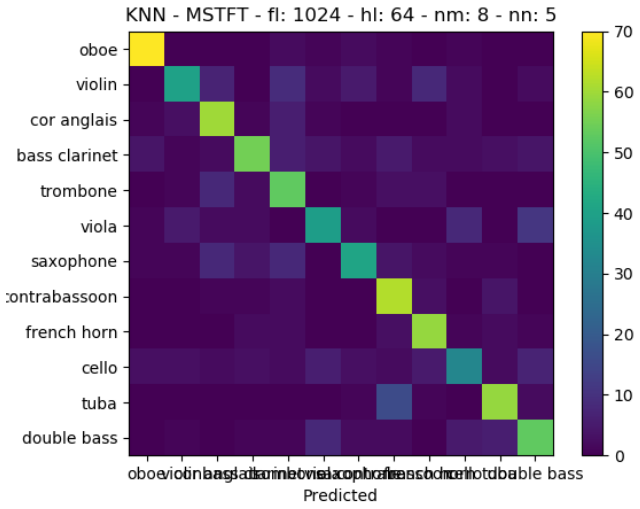

Figure 23: Confusion matrix for KNN + MSTFT (fl = 1024, nm=8)

accuracy: 0.95 | precision: 0.69 | recall: 0.69 | fscore: 0.69

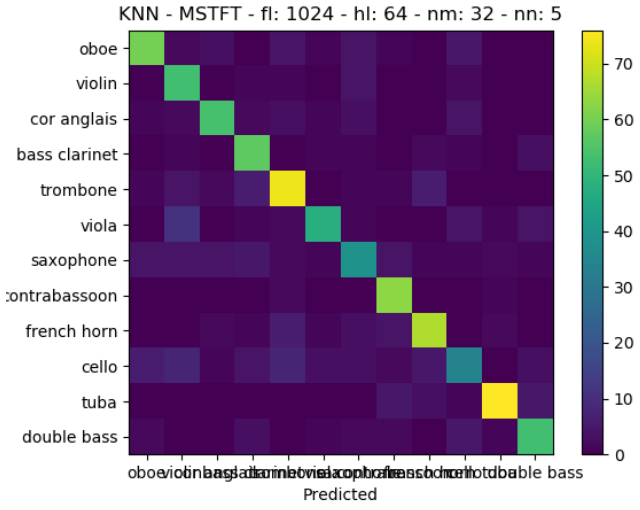

Figure 24: Confusion matrix for KNN + MSTFT (fl = 1024, nm=32)

accuracy: 0.96 | precision: 0.75 | recall: 0.76 | fscore: 0.75

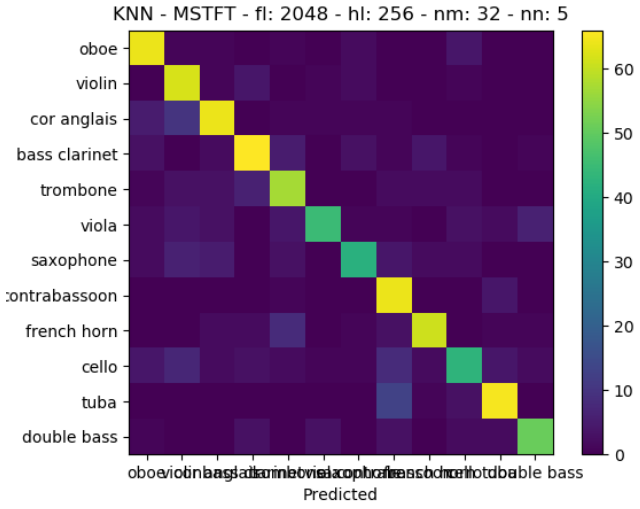

Figure 25: Confusion matrix for KNN + MSTFT (fl = 2048, nm=32)

accuracy: 0.96 | precision: 0.77 | recall: 0.76 | fscore: 0.76

As we can note there's not a big difference between the solutions, that overall are almost good as the STFT one. This makes sense because MSTFT its like a sub-sampled and scaled version of STFT. This solution is so really appreciable because achieves good performances with a smaller size features.

7.1.5 KNN + MFCC

As I discussed in section [4.5], the tuning parameters for this feature are the frame and hop lengths and also the number of mel-bands. Here the results for some different values of them:

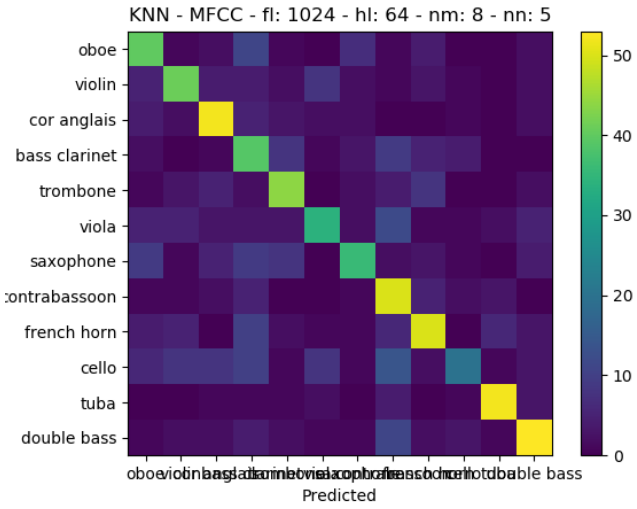

Figure 26: Confusion matrix for KNN + MFCC (fl = 1024, nm=8)

accuracy: 0.93 | precision: 0.58 | recall: 0.57 | fscore: 0.57

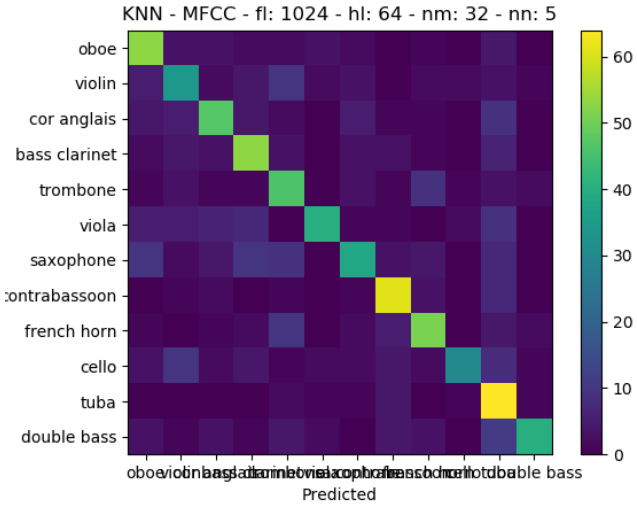

Figure 27: Confusion matrix for KNN + MFCC (fl = 1024, nm=32)

accuracy: 0.94 | precision: 0.65 | recall: 0.62 | fscore: 0.62

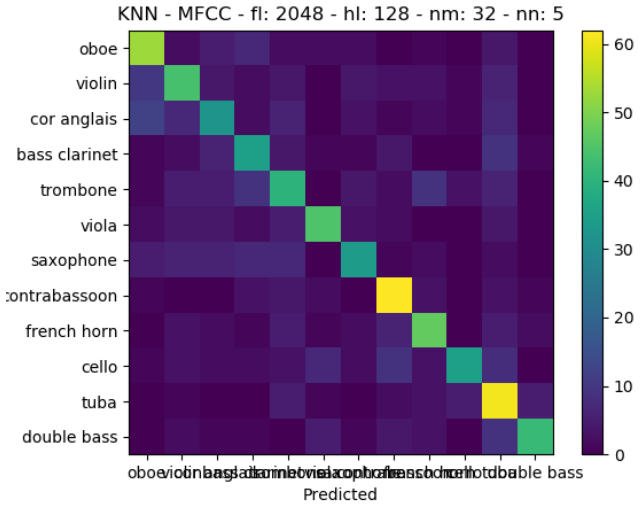

Figure 28: Confusion matrix for KNN + MFCC (fl = 2048, nm=32)

accuracy: 0.93 | precision: 0.61 | recall: 0.59 | fscore: 0.59

As we can note there's not a big difference between the solutions, that overall are slightly worst than the MSTFT one. This solution is anyway appreciable because achieves good performances with very small features.

7.1.6 KNN best model

For KNN the best feature extraction technique result to be MSTFT, and so I perform several test to find out the best value for K. Here I present you some of them by considering a frame length of 2048, hop length of 256 and 32 mel-bands.

k=4 :: accuracy: 0.96 | fscore: 0.76
k=8 :: accuracy: 0.95 | fscore: 0.70
k=12 :: accuracy: 0.95 | fscore: 0.70
k=16 :: accuracy: 0.94 | fscore: 0.66
k=20 :: accuracy: 0.94 | fscore: 0.64

As we can note it turns out that considering a large number of neighbours the performance decreases highlighting that our dataset is really clean (no outliers).

7.2 SVM Results

In order to find the best set up to perform the instrument classification task using SVM, first at all I fix the number of max iteration (mi=10000) and the polynomial order (ord=3) and run some test for each feature. Then, after I found the best one, I try to optimize the SVM parameter such as the type of function to considering during the fitting (kernel trick) and the max iteration value.

As I anticipated you in section 5.2 this technique works only for a one vs one classification task, so in order to perform a multi class problem, n models are individually trained and evaluated. For this reason some of the test I've done had been killed for lack of enough memory resources. In particular of course, this happened when the considered features were very large in terms of dimensionality. That's why in the following discussion I'll introduce the training time as a "metric" to consider while evaluating which model is the best.

Here an example of my resource manager during one of this heavy test.

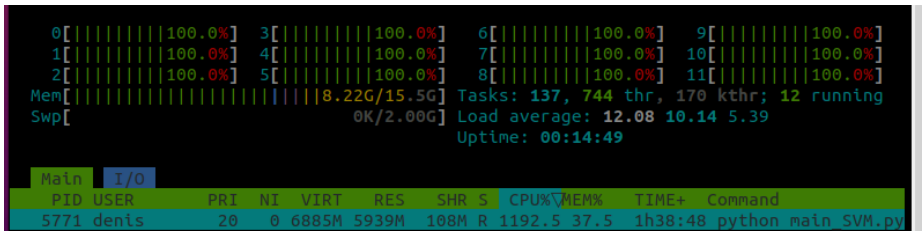

Figure 29: Resource manager during a SVM test with large features

7.2.1 SVM + Signal

As expected and discussed in section [4.1], the original signal is not relevant for this task, and in fact the confusion matrix result not diagonal at all.

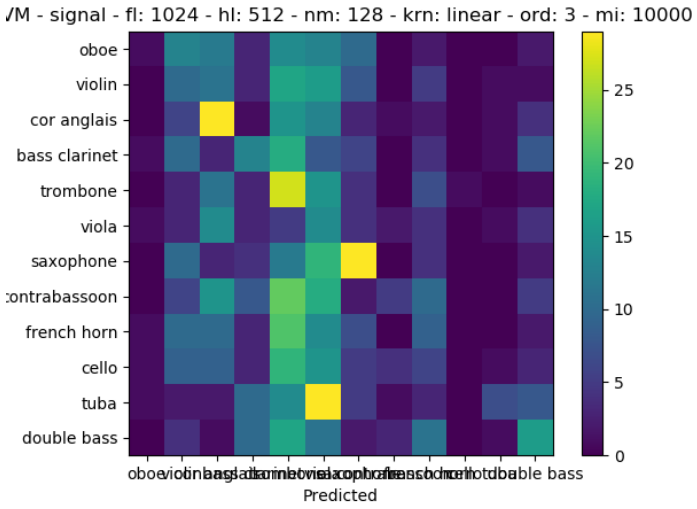

Figure 30: Confusion matrix for SVM + original signal

accuracy: 0.86 | precision: 0.21 | recall: 0.18 | fscore: 0.1

7.2.2 SVM + STFT

As I discussed in section [4.2], the tuning parameters for this feature are the frame and hop lengths. Here the results for some different values of them:

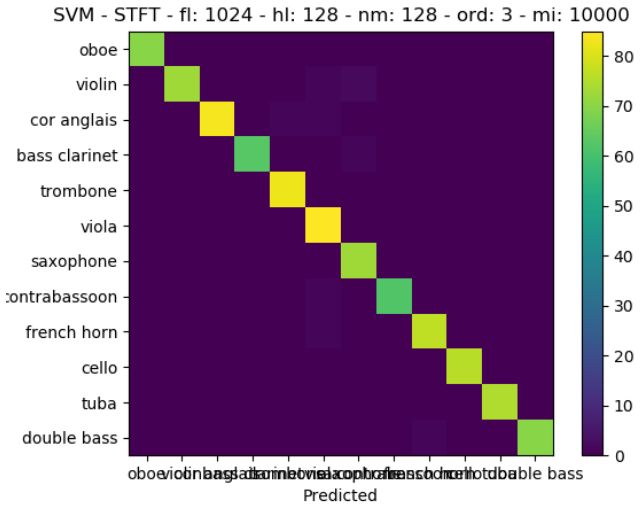

Figure 31: Confusion matrix for SVM + STFT (fl = 1024, hl = 128)

accuracy: 1.0 | precision: 0.99 | recall: 0.99 | fscore: 0.99

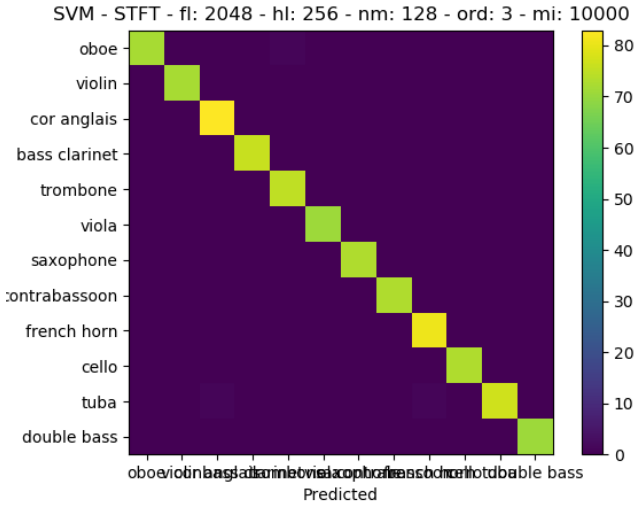

Figure 32: Confusion matrix for SVM + STFT (fl = 2048, hl = 256)

accuracy: 1.0 | precision: 1.0 | recall: 1.0 | fscore: 1.0

As we can note here the results are the ideal ones. Note that STFT produces really big features: 513x173 for frame length 1024 and hop length 128, and 1025x87 for frame length 2048 and hop length 256. For frame length 1024 and hop length 64, that cause 513x345 features, the training cannot end at all in my computer for lack of memory resources. It's also important to note that for this reason the training takes almost one hour for each configuration.

This motivate myself to find the best trade off between speed and performance, and so I tried some other configuration that consider features as small as possible. In particular I increase the frame length and reduce the overlap length to only one quarter of the frame.

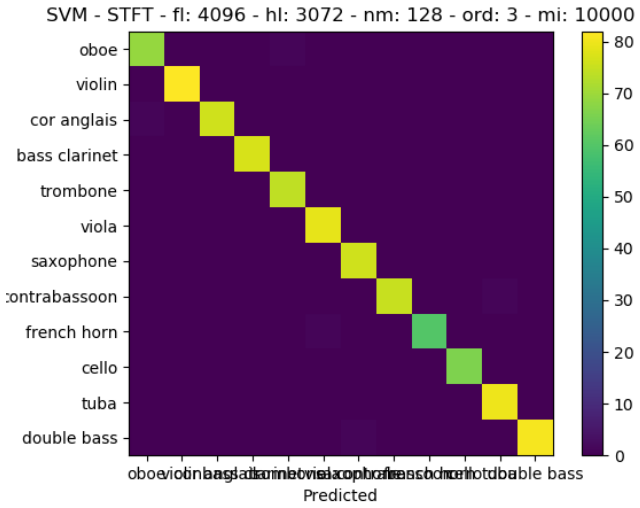

Figure 33: Confusion matrix for SVM + STFT (fl = 8192, hl = 3072)

accuracy: 1.0 | precision: 0.99 | recall: 0.99 | fscore: 0.99

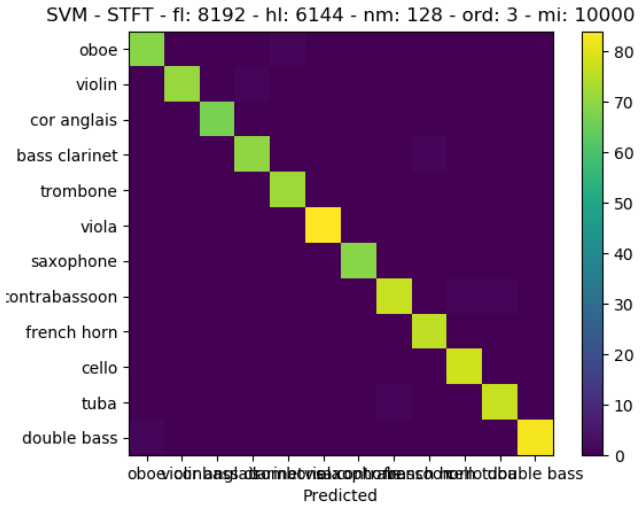

Figure 34: Confusion matrix for SVM + STFT (fl = 8192, hl = 6144)

accuracy: 1.0 | precision: 0.99 | recall: 0.99 | fscore: 0.99

In this cases the training take about 15 minutes, and considering that the performances remain almost identical, this configurations result really appreciable.

7.2.3 SVM + SC

As I discussed in section [4.3], the tuning parameters for this feature are the frame and hop lengths and also the number of bins. Here the results for some different values of them:

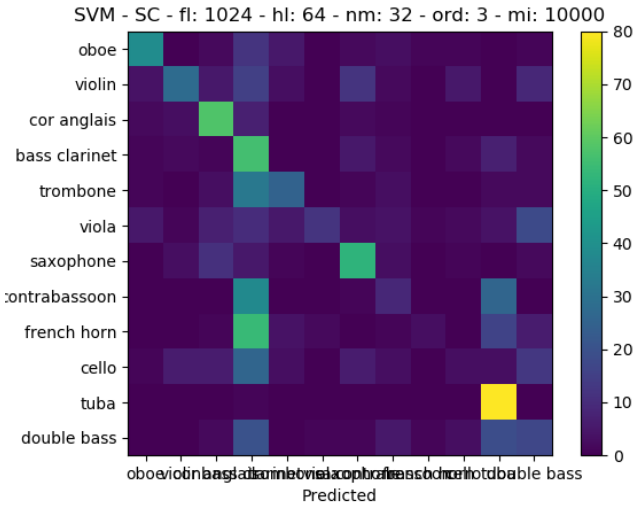

Figure 35: Confusion matrix for SVM + SC (fl = 1024, hl: 64)

accuracy: 0.9 | precision: 0.49 | recall: 0.42 | fscore: 0.39

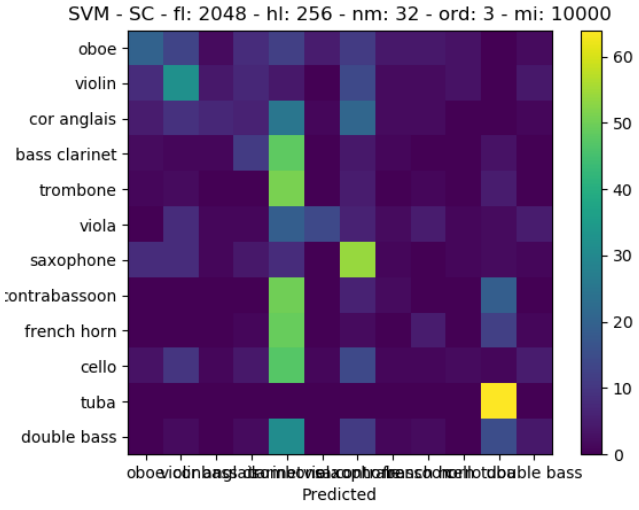

Figure 36: Confusion matrix for SVM + SC (fl = 2048, hl=256)

accuracy: 0.88 | precision: 0.32 | recall: 0.31 | fscore: 0.25

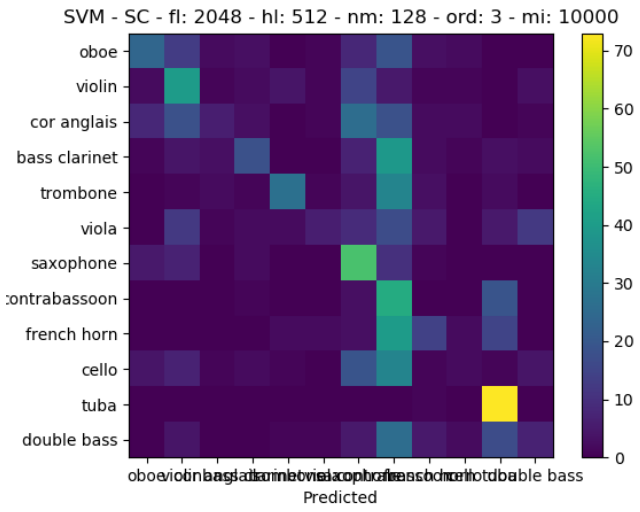

Figure 37: Confusion matrix for SVM + SC (fl = 2048, hl: 512)

accuracy: 0.89 | precision: 0.4 | recall: 0.35 | fscore: 0.31

As we can note for all the different solutions the results are not good as the ones considering spectral centroid but KNN instead of SVM (with same values for all the feature parameters).

In general the performance are not good enough to consider this model in a real application. This bad performance are cause by the feature dimensionality, that being so small helps in terms of training time, but cause a decrease of the performance.

7.2.4 SVM + MSTFT

As I discussed in section [4.4], the tuning parameters for this feature are the frame and hop lengths and also the number of mel-bands. Here the results for some different values of them:

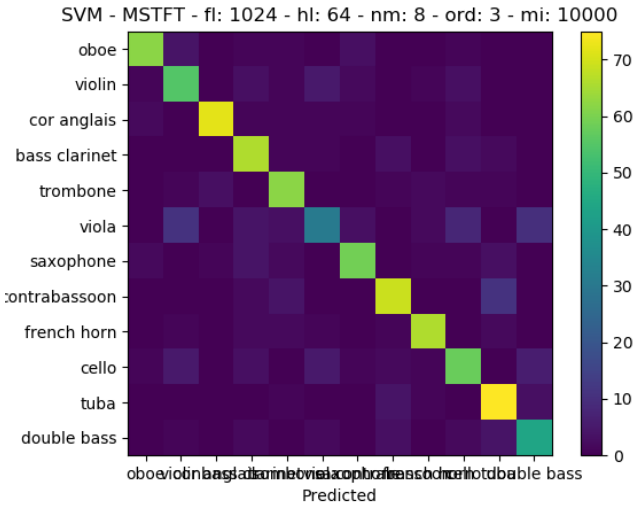

Figure 38: Confusion matrix for SVM + MSTFT (fl = 1024, nm=8)

accuracy: 0.97 | precision: 0.8 | recall: 0.8 | fscore: 0.79

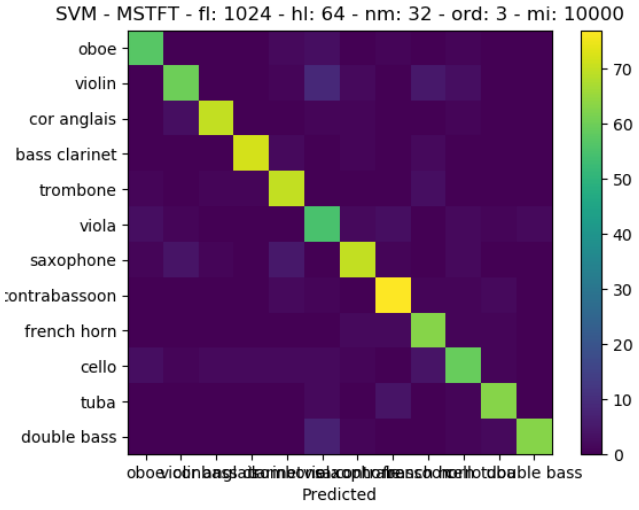

Figure 39: Confusion matrix for SVM + MSTFT (fl = 1024, nm=32)

accuracy: 0.98 | precision: 0.87 | recall: 0.87 | fscore: 0.87

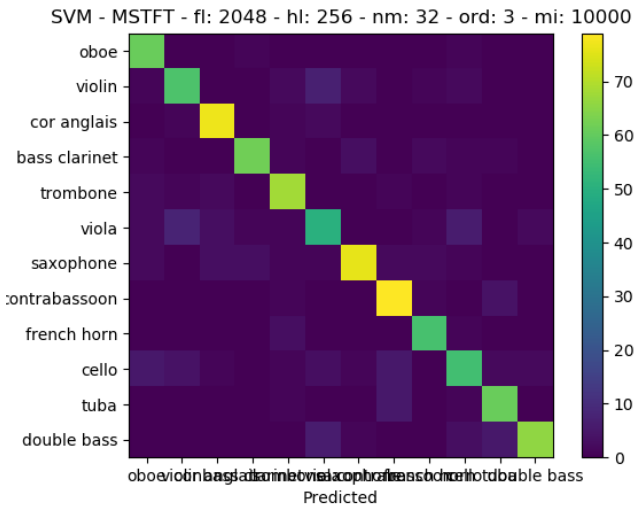

Figure 40: Confusion matrix for SVM + MSTFT (fl = 2048, nm=32)

accuracy: 0.98 | precision: 0.85 | recall: 0.86 | fscore: 0.85

As we can note there's not a big difference between the solutions, that overall are almost really good. In particular the performance are also slightly better that SVM with STFT and KNN with MSTFT, and considering that the training time is not that long, this combination of feature extraction and classification results to be the best one.

7.2.5 SVM + MFCC

As I discussed in section [4.5], the tuning parameters for this feature are the frame and hop lengths and also the number of mel-bands. Here the results for some different values of them:

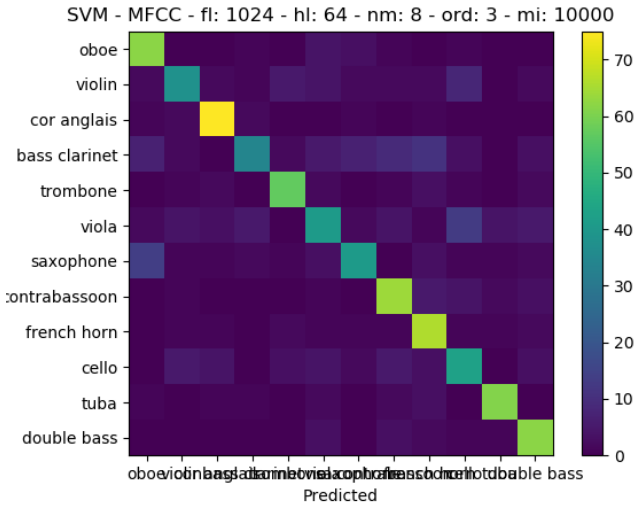

Figure 41: Confusion matrix for SVM + MFCC (fl = 1024, nm=8)

accuracy: 0.95 | precision: 0.72 | recall: 0.72 | fscore: 0.71

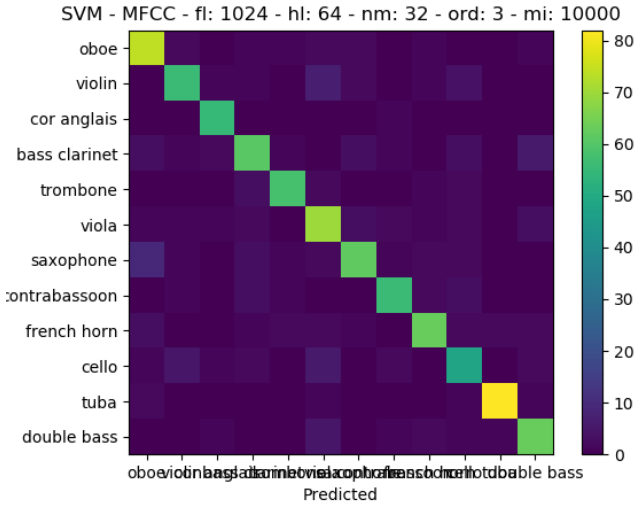

Figure 42: Confusion matrix for SVM + MFCC (fl = 1024, nm=32)

accuracy: 0.97 | precision: 0.83 | recall: 0.83 | fscore: 0.83

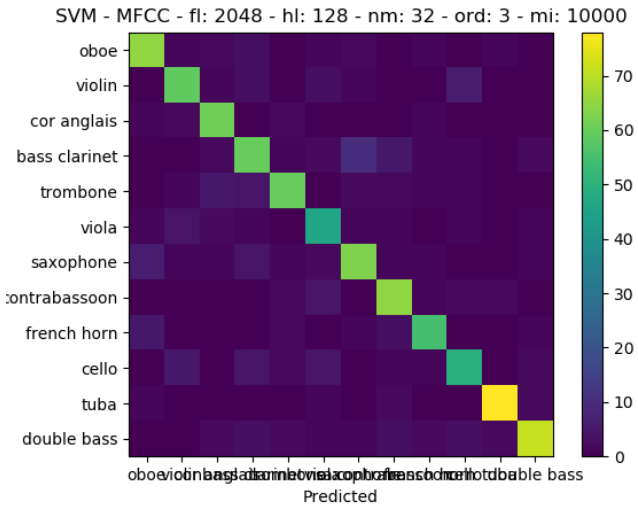

Figure 43: Confusion matrix for SVM + MFCC (fl = 2048, nm=32)

accuracy: 0.97 | precision: 0.81 | recall: 0.81 | fscore: 0.81

As we can note there's not a big difference between the solutions, that overall are good. Anyway comparing this solution with the MSTFT one, it results less appreciable because the performances are slightly worst and the training takes much more time. Instead this solution result a bit better compared to the KNN and MSTFT considering the same parameters.

7.2.6 SVM best kernel

For SVM the best feature extraction technique results to be MSTFT, and so I perform several test to find out the best value for the kernel order. Here I present you some of them by considering a frame length of 1024, hop length of 128 and 32 mel-bands.

order=1 :: accuracy: 0.96 | fscore: 0.76
order=3 :: accuracy: 0.98 | fscore: 0.86
order=5 :: accuracy: 0.98 | fscore: 0.9
order=7 :: accuracy: 0.98 | fscore: 0.88
order=9 :: accuracy: 0.98 | fscore: 0.88
order=11 :: accuracy: 0.98 | fscore: 0.87

As we can note it turns out that the best kernel is the polynomial one of order 5, and then for higher order the performance remain almost the same.

7.2.7 SVM best max iteration

For SVM the best feature extraction technique result to be MSTFT, and so I perform several test to find out the best value for the max iteration. Here I present you some of them by considering a frame length of 2048, hop length of 256 and 32 mel-bands.

```
max_i=100   :: accuracy: 0.96 | fscore: 0.76
max_i=1000  :: accuracy: 0.95 | fscore: 0.70
max_i=10000 :: accuracy: 0.95 | fscore: 0.70
```

As we can note it turns out that increasing the number of max iteration the performances remain almost the same. Considering that a big value of the max iteration cause a longer training time, a big value of it is not recommended.

8 Conclusion

The presented solutions result to be all good for the considered task, but some of them are more interesting for some particular aspect than others. In particular KNN result really appreciable for its speed, and being one of the simplest classification technique at all, highlights the goodness of the considered dataset and feature extraction techniques. For a generic application KNN results to be the best choice in terms of classification. On the other hand SVM results much more slow during training time, but achieving really surprising performance, it can be considered the best choice in any application that doesn't require a fast response.

Overall the worst feature for this task results to be the spectral centroid, and the best one the MSTFT because it achieve both small features and high performances. For the classification part the best model results to be SVM with a 5th order polynomial kernel, but it need a careful choice on the feature to consider with it.

References

[1] Musical Sound, Instruments, and Equipment. https://iopscience.iop.org/book/mono/978-1-6817-4680-7.pdf

[2] Musical genre classification of audio signals. https://ieeexplore.ieee.org/document/1021072

[3] Music Emotion Recognition: A State of the Art Review. https://musicmachinery.com/2010/08/11/music-emotion-recognition-a-state-of-the-art-review/

[4] Automatic Classification of Musical Instrument Sounds. https://www.tandfonline.com/doi/abs/10.1076/jnmr.32.1.3.16798

[5] Social Tagging and Music Information Retrieval. https://musicmachinery.com/2009/05/11/social-tags-and-music-information-retrieval/

[6] Speech Recognition. https://en.wikipedia.org/wiki/Speech_recognition

[7] Filtering Techniques for Noise Reduction and Speech Enhancement. https://link.springer.com/chapter/10.1007/978-3-662-11028-7_5

[8] A sound based method for fault detection with statistical feature extraction in UAV motors. https://www.sciencedirect.com/science/article/abs/pii/S0003682X21004199

[9] A programming language that lets you work quickly and integrate systems more effectively. https://www.python.org/

[10] Simple and efficient tools for predictive data analysis. https://scikit-learn.org/stable/

[11] Audio and music processing in Python. https://librosa.org/

[12] Visualization with Python. https://matplotlib.org/

[13] Download thousands of free sound samples recorded by Philharmonia musicians. https://philharmonia.co.uk/resources/sound-samples/