CLASSIFICATION OF INDUSTRIAL MACHINES SOUNDS

Università degli Studi di Milano, "Audio Pattern Recognition" course

Marco Cavallari

***Abstract*—Factory machinery is prone to failure or breakdown, resulting in significant expenses for companies. Hence, there is a rising interest in machine monitoring using different sensors including microphones [1]. In this paper, I will present some techniques of classification as Neural Network and SVM that allow to understand if a particular mechanical machine is malfunctioning or not. In order to do this, It has been used a sub-portion of the MIMII dataset. In MIMII dataset, normal sounds were recorded for different types of industrial machines (i.e., valves, pumps, fans, and slide rails), and to resemble a real-life scenario, various anomalous sounds were recorded (e.g., contamination, leakage, rotating unbalance, and rail damage) [1]. The results achieved in this paper are satisfactory: both NNs and SVMs were able to identify normal and abnormal sounds correctly.**

I. INTRODUCTION

In the past decade, industrial Internet of Things (IoT) and data-driven techniques have been revolutionizing the manufacturing industry, and different approaches have been undertaken for monitoring the state of machinery. Examples include vibration sensor-based approaches, temperature sensor-based approaches, and pressure sensor-based approaches [1]. Another important approach that has been addressed in this paper is the use of technologies for acoustic classification in order to detect anomalies. What has been done here is a simple classification of sounds of mechanical parts. To do this, the MIMII dataset has been used. This dataset contains sounds of different types of machines: valves, pumps, fans, slide rails. Each machine may be part of a different product model (numerated by 00 to 06). For each machine type are present some audio that identifies normal behaviors and some other that identify anomalies. In this project, It has been analyzed just the pumps and the fans audio of model 02. This because the entire dataset was too large to be processed on my machine. What has been done is develop some classification models as NNs and SVMs, train them with a portion of data, validate them and then use these models in order to predict the class of a third portion of the dataset not used for training and validating: the test-set. We will discuss more in detail further. There are other works in this field, in particular, an experiment was proposed by the "Research and Development Group, Hitachi, Ltd". In this work, more exhaustive results were achieved and the complete MIMII dataset was used.

Table 1: MIMII dataset content details.

Machine type / model ID		Segments for normal condition	Segments for anomalous condition
Valve	00	991	119
	01	869	120
	02	708	120
	03	963	120
	04	1000	120
	05	999	400
	06	992	120
Pump	00	1006	143
	01	1003	116
	02	1005	111
	03	706	113
	04	702	100
	05	1008	248
	06	1036	102
Fan	00	1011	407
	01	1034	407
	02	1016	359
	03	1012	358
	04	1033	348
	05	1109	349
	06	1015	361
Slide rail	00	1068	356
	01	1068	178
	02	1068	267
	03	1068	178
	04	534	178
	05	534	178
	06	534	89
Total		26092	6065

Fig. 1. MIMII dataset structure and composition

II. METHODS

A. 10-FOLD-CROSS VALIDATION

Cross-validation is a resampling procedure used to evaluate machine models learning on a limited sample of data. You just need to specify a single parameter k which represents the number of groups (also called folds) in which a given sample of data needs to be split. These groups must contain approximately the same number of examples. The algorithm of the k-Fold technique works as follows:

1) A fold number (k) is chosen. Usually, k is 5 or 10,
2) The data set is partitioned into k equal parts (if possible),

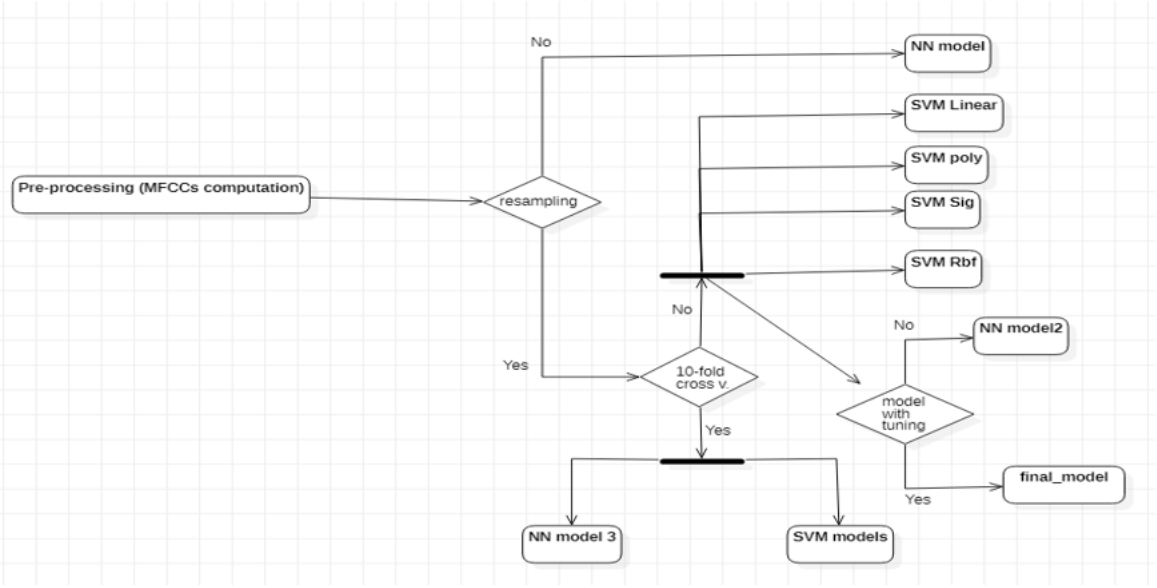

Fig. 2. Block diagram of the methods used

3) k - 1 fold will constitute the training set. The remaining group will be the test set,
4) The model is trained on the training set. At each iteration of cross-validation, A new model must be trained independently from the trained model in the previous iteration
5) Validation is carried out on the test set,
6) The validation result is saved,
7) Steps 3 to 6 are repeated k times (each time the fold that constitutes the test set will be different). In this way you will have validated the model on each fold of the dataset,
8) To obtain the final score, the average of the results obtained in step 6 is carried out.

This method allows you to divide the sample in such a way that each fold has a good one representation of the entire data set. What we are particularly interested in is that the data of the classes existing are split equally in the different folds

B. NEURAL NETWORK

In this project I have used a feed forward neural network. It consists of three types of layers—the input layer, output layer and hidden layer. The input layer receives the input signal to be processed. The required task such as classification is performed by the output layer. An arbitrary number of hidden layers that are placed in between the input and output layer are the true computational engine of the Network[3]. With this model the data flows in the forward direction from input to output layer. In this particular model I have used three type of layers:

1) Dense: Neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer.
2) Activation: The activation function initially choosen was ReLU. The rectified linear activation function or ReLU is a linear function that will output the input directly if it is positive, otherwise, it will output zero. It can be seen as a function:

$$f(x) = max(0, x) \tag{1}$$

3) Dropout: Dropout is a technique used to prevent a model from overfitting. Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase. This prevents units from co-adapting too much.So, if you set half of the activations of a layer to zero, the neural network won't be able to rely on particular activations in a given feed-forward pass during training. As a consequence, the neural network will learn different, redundant representations.

C. SVM

The Support Vector Machine algorithm is used in supervised machine learning to solve classification and regression problems. The training process receives a dataset as input. Each row of the dataset is characterized by X features and a label.

For each element of the dataset, the set of its characteristics identifies a point in space (with n dimensions where n is the number of features). The vector y contains the labels (label) that is the correct classification of each example.

The goal of the algorithm is to find a hyperplane capable of dividing the data set into k classes where k is the number of labels.

The boundary that separates the classes is called the decision boundary.

Points closest to the hyperplane (support vector) are more prone to adding new data. A small variation of the hyperplane can change their classification. The distance between the support vectors and the decision boundary is called margin and the goal is to maximize it.

The hyperplane is identified by the equation

$$w * x - b = 0. \tag{2}$$

The SVM algorithm finds the optimal values of w and b by solving an optimization problem during the training process.

Once found, it uses them in the sign function :

$$y = sign(wx - b) \tag{3}$$

to classify the data.

The y value takes on these values +1 if the function $w * x - b \geq 0$ and -1 if the function w * x-b ≤ 0

What if decision boundary is not linear? We can use the kernel trick. The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined.

D. SMOTE

SMOTE (or Minority Oversampling TEchnique) randomly selects an instance of the less represented class "A" and finds its closest k neighbors (of the same class). The synthetic instance is then created by choosing one of the closest k neighbors "B" at random and a synthetic example is created at a randomly selected point between the two examples in feature space[4].

III. APPLICATION OF METHODS AND RESULTS

What has been done is straightforward. First, the Mfcc of all the audio has been computed. Then a first model (the "NN model" in Fig. 2) has been trained, validated and tested. The accuracy of the model was quite good (91.7%) but there was a significant problem (as we can also see at Fig. 3): since the class "Abnormal pump" and "Abnormal fan" were too less represented (351 and 109 audios) in the dataset in comparison with the "Normal" classes (1006 for the "Normal fan" class and 1001 for the "Normal pump" class), the model was not able to recognize the audio of those classes. For this reason, the model has been resampled via SMOTE. After the resampling, a new model ("NN model2") has been created (NOTE: both "NN model2" and "NN model" share the same parameters). The results of this second model (Fig. 4 and Table 1), as expected, were better than the previous ones: both accuracy, f1 score, recall and precision were around 94.7%.

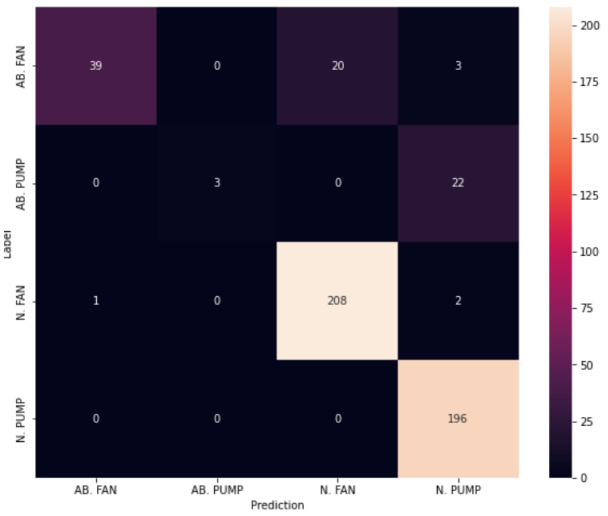

Fig. 3. Confusion matrix of "NN model"

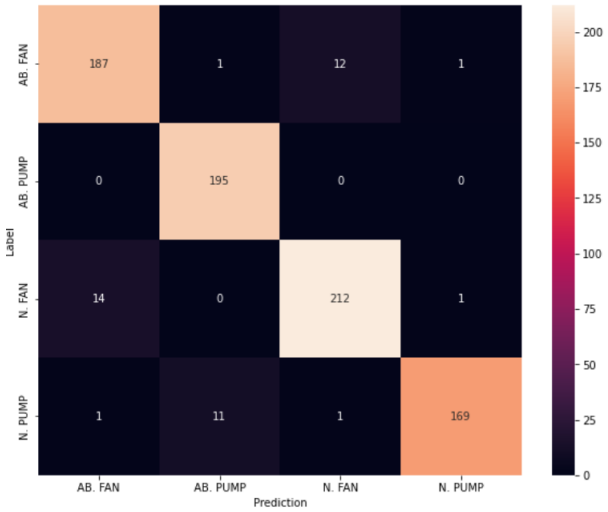

Fig. 4. Confusion matrix of "NN model2"

Metric	% (0 to 1)
Accuracy	0.9478
F1 Score	0.9477
Precision	0.9485
Recall	0.9478

TABLE I
METRICS OF "NN MODEL2"

Since the goal is to always improve our models metrics, what now we did was to try to use 10-fold cross-validation. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. For this reason, the "NN model 3" has been developed. The results (Table 2) have not improved: a 93.3% of accuracy, precision, recall and f1 score have been recorded, so less than the previous model. The last experiment I did about NN was

Metric	% (0 to 1)
Accuracy	0.9338
F1 Score	0.9336
Precision	0.9374
Recall	0.9338

TABLE II
METRICS OF "NN MODEL3"

to try to improve the model tuning the fundamental parameters of it such as:

1) Epochs: number that defines the number of times that the learning algorithm will work through the entire training dataset.The best results were achieved with 200 as epochs number.
2) Batch size: number of samples to work through before updating the internal model parameters. The best results were achieved with 40 as batch size.

1. Best: 0.950640 using 'batch$_size'$: 40,$'epochs'$: 200
2. 0.593742 (0.049478) with: 'batch$_size'$: 12,$'epochs'$: 10
3. 0.797338 (0.079484) with: 'batch$_size'$: 12,$'epochs'$: 50
4. 0.908533 (0.017383) with: 'batch$_size'$: 12,$'epochs'$: 100
5. 0.889892 (0.012785) with: 'batch$_size'$: 12,$'epochs'$: 150
6. 0.849486 (0.056654) with: 'batch$_size'$: 12,$'epochs'$: 200
...
...

Activation function: defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network. The best results were achieved with "SoftPlus" or "SoftSign" as activation function for all the layers except from the last one (where "SoftMax" was used).

1. Best: 0.940979 using 'activation': 'softsign', 'final$_activation'$:$'softmax'$
2. 0.489133 (0.012703) with: 'activation': 'softmax', 'final$_activation'$:$'softmax'$
3. 0.319706 (0.132672) with: 'activation': 'softmax', 'final$_activation'$:$'softplus'$
4. 0.243363 (0.022249) with: 'activation': 'softmax', 'final$_activation'$:$'softsign'$
5. 0.255090 (0.007022) with: 'activation': 'softmax', 'final$_activation'$:$'relu'$
6. 0.253370 (0.011138) with: 'activation': 'softmax', 'final$_activation'$:$'tanh'$
...
...

Optimizers: they are algorithms or methods used to change the attributes of the neural network such as weights and learning rate to reduce the losses. The best results were achieved with "Nadam", "Adam", "RMSprop" or "Adamax" as optimization algorithm.

1. Best: 0.945808 using 'optimizer': 'Nadam'
2. 0.234041 (0.015326) with: 'optimizer': 'SGD'
3. 0.920965 (0.029666) with: 'optimizer': 'RMSprop'
4. 0.288206 (0.051691) with: 'optimizer': 'Adagrad'
5. 0.240945 (0.019892) with: 'optimizer': 'Adadelta'
6. 0.925785 (0.005172) with: 'optimizer': 'Adam'
7. 0.906119 (0.020681) with: 'optimizer': 'Adamax'
8. 0.945808 (0.005074) with: 'optimizer': 'Nadam'

After the tuning, a final model (that produced the results at Fig. 5 and Table 3) has been created with the parameters found at the previous step. Great results were recorded: 97% in all the metrics taken into account.

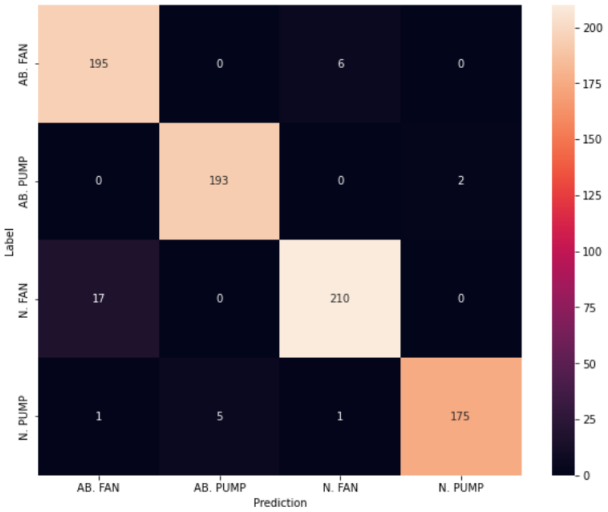

Fig.5.Confusion matrix of "final model"

Metric	% (0 to 1)
Accuracy	0.9602
F1 Score	0.9603
Precision	0.9611
Recall	0.9603

TABLE III
METRICS OF "FINAL MODEL"

In addition to Neural Networks also SVMs have been used to classify industrial machine sounds. With this classifier, We have used the same train, validation and test sets in order to be able to compare the results with the NNs seen before. First we analyzed the different SVMs:

1) One with a linear kernel (results on test set at Fig. 6 and Table 4): It is the only one that gave good results. After the training, it was then evaluated on the validation set and it achieved a 96% of accuracy.

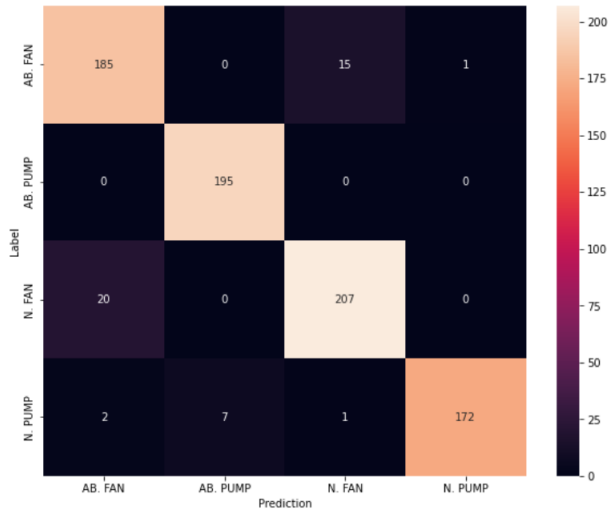

Fig.6.Confusion matrix of ”SVM linear”

2) One with a radial basis function as kernel (results on test set at Fig. 7 and Table 4): Same steps as before (training and validation) but worst results: 49% of accuracy.

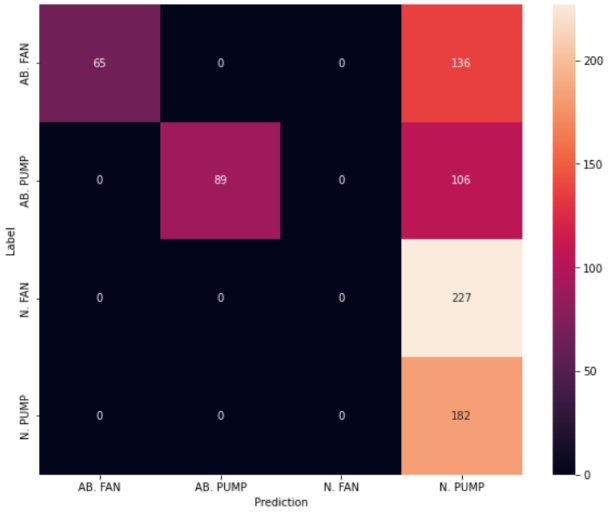

Fig.7.Confusion matrix of ”SVM Rbf”

3) One with polynomial kernel (results on test set at Fig. 8 and Table 4): 68% of accuracy.

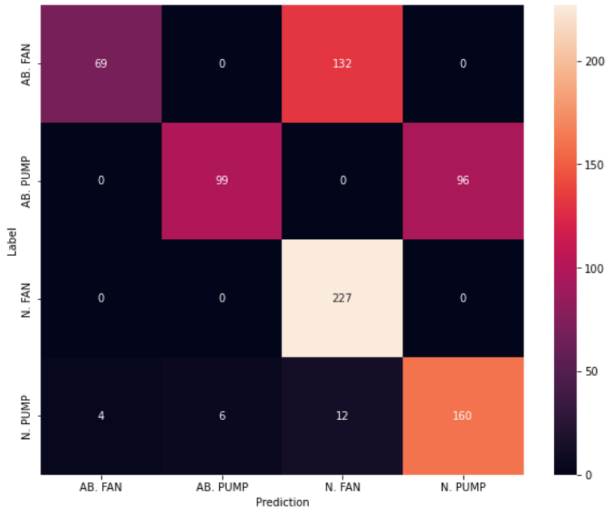

Fig.8.Confusion matrix of ”SVM poly”

4) One with a sigmoid kernel (results on test set at Fig. 9 and Table 4): 39% of accuracy.

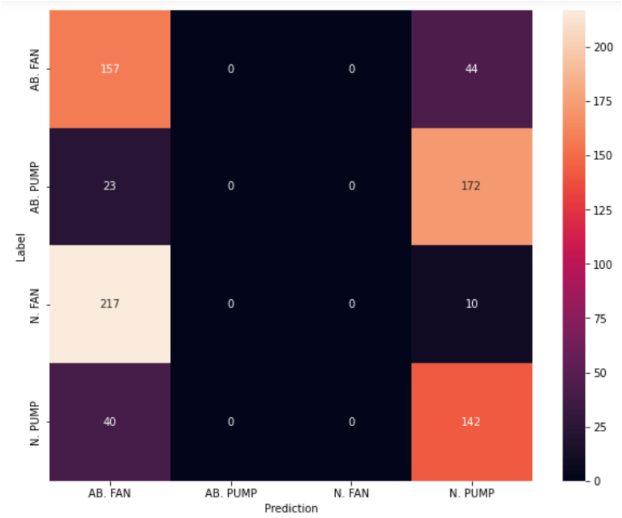

Fig.9.Confusion matrix of ”SVM Sig”

In Table 4 you can observe the metric results of he various SVMs.

As a final step of this project, what I did was to try to use 10-fold cross-validation but as with Neural Networks, the results achieved with it did not improve our previous model. In Table 5 We can see the results.

IV. CONCLUSION

The performances of both neural network and SVM (with the linear kernel) are satisfactory. Maybe the neural network (with hyperparameters tuning) has better results (95/96 % of accuracy, F1 score etc..) Respect to the ”linear SVM” (94%).

Kernel	Accuracy	F1 Score	Precision	Recall
Linear K, Model	0.9429	0.9429	0.9435	0.9429
Radial Basis K. Model	0.4174	0.3726	0.5551	0.4174
Polynomial K. Model	0.6894	0.6649	0.7782	0.6894
Sigmoid K. Model	0.3714	0.2396	0.1770	0.3714

TABLE IV
METRICS OF THE SVMS

Kernel	Accuracy	F1 Score	Precision	Recall
Linear K, Model	0.9391	0.9392	0.9399	0.9491
Radial Basis K. Model	0.4286	0.3851	0.5561	0.4286
Polynomial K. Model	0.6981	0.6749	0.7799	0.6981
Sigmoid K. Model	0.3453	0.2226	0.1715	0.3453

TABLE V
METRICS OF THE SVMS WITH 10-FOLD-CROSS VALIDATION

Unfortunately is quite difficult to compare the results of my work to existing ones since the I used in my project a smaller dataset composed by sounds of different machine types from two dataset: "-6_dB_fan.zip" and "-6_dB_pump.zip".Both of them contained normal and abnormal sound of 6 different machines; I analyzed just the ones of the "machine 2". In addition to this, the other works considered different levels of SNR (with factory noise): for example, 6 dB, 0 dB, and 6 dB. I just used one of them. To resume, I just used two kind of sounds (fan and pump) of one particular machine (machine 2).

In conclusion, I will put here a Table that will the F1 Score of all the model with the different configurations used.

NB: the results reported on this paper and the ones in the code could be slightly different due to the possibility of coming from different experiment (different execution)

Model	Description	F1 Score
NN model	Initial model with no resampled data and no tuning of the Network	0.9016
NN model 2	Model with resmapled data and no tuning of the Network	**0.9477**
NN model 3	Model with 10-fold cross validation and resampled data (no tuning)	**0.9336**
final model	Model with resampled data and tuning of the Network	**0.9603**
SVM Linear	SVM with linear kernel (resampled data)	**0.9429**
SVM poly	SVM with polynomial kernel	0.6649
SVM Sig	SVM with sigmoid kernel	0.2396
SVM Rbf	SVM with Radial Basis Function Kernel	0.3727
SVM Linear(10)	SVM with linear kernel (resampled data) and 10-fold-cross validation	**0.9392**
SVM poly(10)	SVM with polynomial kernel (resampled data) and 10-fold-cross validation	0.6749
SVM Sig(10)	SVM with sigmoid kernel (resampled data) and 10-fold-cross validation	0.2226
SVM Rbf(10)	SVM with Radial Basis Function Kernel (resampled data) and 10-fold-cross validation	0.3851

In addition to this, from what I am able to understand by their paper, They analyzed Fans and Pumps (and other type I did not considered) in an independent way, so their work was just to determine if a fan was normal or not. In my work pumps and fans sounds were mixed in a single dataset.

All of this made a comparison quite useless since also the dataset used was completely different

REFERENCES

[1] Harsh Purohit, Ryo Tanabe, Kenji Ichige, Takashi Endo, Yuki Nikaido, Kaori Suefusa, Yohei Kawaguchi, "TMIMII Dataset: Sound Dataset for Malfunctioning Industrial Machine Investigation and Inspection"Cornell University, 20 Sep 2019.

[2] "How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras" by Jason Brownlee on August 9, 2016

[3] "The Digital Twin Paradigm for Smarter Systems and Environments: The Industry Use Cases" by S. Abirami, P. Chitra, in Advances in Computers, 2020

[4] SMOTE for Imbalanced Classification with Python by Jason Brownlee on January 17, 2020