



Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg



BACHELOR THESIS

YEAR 2018-2019

MACHINE LEARNING FOR NOISE REDUCTION IN IMAGES OF OLD AUDIO RECORDS

Report

Benoit RUFFRAY

Date : 09/08/2019

External supervisor: Haber Carl

Internal supervisors: Bapst Frédéric / Hennebert Jean

Experts: Grisanti Vito / Mosanya Emeka

Consultants: Cornell Earl / Nachman Benjamin

Contents

1	Introduction	2
1.1	Report organization	2
1.2	Folder organization	2
2	Context	3
2.1	Mechanical Audio records	3
2.2	IRENE	4
2.3	Weaver	6
2.4	Machine learning and deep neural networks	6
2.5	Shellac disc images properties	7
2.6	Audio data	7
3	Objectives	7
3.1	Prototype 0	7
3.2	Prototype 1	9
3.3	Weaver Plugin	9
3.4	Prototype 2	9
3.5	Prototype 3	9
3.6	Prototype 4	9
4	Tasks	10
4.1	Prototype 0	10
4.2	Prototype 1	10
4.3	Weaver Plugin	11
4.4	Prototype 2	11
4.5	Prototype 3	11
4.6	Prototype 4	12
4.7	Integration to Weaver	12
4.8	Planning	12
4.9	Risks	12
5	Prototype 0	12
5.1	Keras and TensorFlow	13
5.2	CIFAR-10	13
5.3	Convolutional Neural Networks	14
5.4	Implementation and results	14

6	Prototype 1	14
6.1	SOTA machine learning models	14
6.2	Risks	17
6.3	Weaver plugins	17
6.4	Dataset generation plugin	17
6.5	Dataset	18
6.6	Model implementation	21
6.6.1	Version 1	21
6.6.2	Version 1 results	21
6.6.3	Version 2	21
6.6.4	Version 2 - results	24
6.6.5	Version 3	25
6.6.6	Version 3 - results	28
6.6.7	Version 4	28
6.6.8	Version 4 results	32
7	Prototype 2	33
7.1	Noisy groove images	35
7.2	Artificially noised dataset	35
7.3	Version 1	35
7.3.1	Implementation	38
7.3.2	Results	38
7.4	Version 2	39
7.4.1	Implementation	39
7.4.2	Results	39
8	Prototype 3 and 4	40
8.1	Dataset	40
8.2	Direct prediction	47
8.3	Transfer learning	50
8.4	Training new model	54
9	Conclusion	54
9.1	Personal conclusion	55
9.2	Future work	55
10	Declaration of honor	56
11	References	56

Appendices	56
A Prototype 1	56
A.1 Models and results	56
A.2 Image dataset	59
B Prototype 2	62
B.1 Models and results	62
B.2 Image dataset	63
C Prototype 3 and 4	68
C.1 Models and results	68
C.2 Image dataset	69
D Planning	72

Abstract

Audio recording has a long history. It started in 1877 with Thomas Edison's phonograph, able to physically record sounds on cylinders made out of wax or tinfoil and to play them back. Those materials are brittle, and they wear out with each play back. Over the years, multiple improvements were made on the concept, changing the records' shape and material, until the disc we know and use. However, it is those old brittle records that interest us.

The Lawrence Berkeley National Laboratory is home of an important historical project : the preservation of old audio records. These cylinders and discs contain historical data such as presidential speeches, native American interviews, traditional songs, and so on. Since they wear out when played physically, they had to find a way to read them without touching them.

IRENE is an imaging machine able to digitize any audio support with high resolution 2D and 3D cameras. Having high fidelity pictures of the discs and cylinders assures preservation. They then started to search for ways to play back those records using only the images.

The team at the Laboratory uses a deterministic algorithm to detect the groove edges, simulate the needle following it, and compute the resulting sound wave. However defects, scratches, dust, and even the texture add a noise on the image that is also detected by their algorithm. This results in a high-pitched background noise when simulating a play back.

In recent years, machine learning has shown impressive results in a large variety of tasks, and can certainly help here. The idea of ML4NR project is to train a model to generate a sound wave by only looking at the disc image. This is developed in multiple steps : - Step 1 : train on clean groove images artificially generated with clear sound as goal. - Step 2 : train on noisy artificial groove images, still using the clear sound as a goal. - Step 3 : adapt to the disc images, train more if necessary. A big part of the project is also the auto-formation in machine learning, as we haven't practiced it a lot.

We mainly focused on one type of neural network in this project : convolutional neural networks. Their advantage is the ability to learn shapes and features at different scales, which seems good for our problem. The main challenge is finding the right parameters for training : what size should the convolution filters be, how many layers of them, do we use dropout or not, etc. Deep learning is still a dark art, and the only way to find the right combination of parameters is to try.

Our convolutional network shows promising results in steps 1 and 2, generating a sound wave close to the truth even when the image is noisy (black rectangles hiding things, Gaussian noise). However, we couldn't find the right combination of parameters for actual disc images, and the generated audio is

mainly background noise. This still open doors for future work in the same direction, as the input data of step 2 isn't so different from step 3.

Acknowledgements

I would like to thank Carl Haber, the Lawrence Berkeley National Laboratory, and the College of Engineering and Architecture of Fribourg for giving me the opportunity to do my Bachelor thesis in California.

My thanks also go to Frédéric Bapst, Jean Hennebert, Vito Grisanti, and Emeka Mosanya, whose supervision and help made this project possible.

Finally I would like to thank Earl Cornell, Benjamin Nachmann, and Carl Haber again, for their continuous support at the Laboratory during the project.

1 Introduction

This document is the final report of the Bachelor project "Machine learning for noise reduction in images of old audio records". It contains all information regarding the project, from the context to the details of implementation. This project is done under the supervision of Mr. Carl Haber at the Lawrence Berkeley National Laboratory, and supervised by Messrs. Frédéric Bapst and Jean Hennebert at the University of Applied Science of Fribourg.

"Machine learning for noise reduction in images of old audio records" (ML4NR) is a project proposed by Mr. Carl Haber at the Lawrence Berkeley National Laboratory (LBNL). It is part of a project for the Library of Congress, whose goal is the restoration and preservation of old audio records. ML4NR is done as a Bachelor Thesis project by Mr. Benoît Ruffray

1.1 Report organization

This report starts with an explanation of the context regarding audio records, IRENE/Weaver, and machine learning. It continues by detailing the objectives of the project and the tasks planning. Then, each major step is explained in its own section, from the analysis to the testing. Finally, the conclusion and personal review resumes it all.

The appendices contain details on dataset properties, models parameters and architecture, and all results obtained during training and testing.

1.2 Folder organization

The git project folder contains a `doc` folder, a `code` folder and a `READ-ME` file. The `doc` folder contains the following elements:

data This folder contains everything regarding the properties of the data used and the system's architecture.

images This folder contains the images used for the different documents.

planning This folder contains the files used for Gantt planning.

pv This folder contains all the meeting minutes.

report This folder contains the report document.

specifications This folder contains the specification document.

The **code** folder contains the following elements:

Jupyter notebooks A Jupyter notebook for each prototype, containing an autonomous code cell for each model and algorithm used.

The code developed for Weaver is only available on its GitHub repository. The disc images and music can be found in the Google Drive folder given in the **READ-ME** file, as they take a lot of space.

2 Context

2.1 Mechanical Audio records

The first device able to record and play back sound waves was an invention of Thomas Edison in 1877, the cylinder phonograph[1]. This device could record sounds by making a needle vibrate and carve a groove on a rotating tinfoil or wax cylinder. To play it back, the cylinder was rotated, making the needle vibrate while following the groove. This vibration is amplified and produces audible sound waves, the ones recorded.

Since cylinders were impractical to store and quick to be damaged (the physical contact would wear the groove out during each play), Emile Berliner improved the concept in 1887 by inventing a flat support for audio recording: discs. Those were easier to stack and reproduce, and therefore quickly overtook the market. Their associated reading device is the gramophone.

The standard format from the 1910s to 1950s was a double-sided 78 rpm shellac disc. However, shellac is a brittle material, and the hard needles would wear them out quickly. It's in 1948 that Columbia Records invented the 33 rpm vinyl disc. Vinyl is more expensive but sturdier than shellac, and the longer playtime would compensate

for the extra cost. RCA Victor introduced the 45 rpm, smaller vinyl disc, in 1949, effectively cutting the cost by using less material. Both formats replaced the shellac disc.

The Library of Congress is interested in the preservation of these old records, as they contain valuable historical audio data such as presidential speeches, native American interviews, or traditional songs from the past. However, since the cylinders and shellac discs are getting destroyed with each play, they associated with the LBNL to find a way to extract and preserve all data without physical contact (non invasive play back). This collaboration gave birth to IRENE.

2.2 IRENE

Stands for Image, Reconstruct, Erase Noise, Etc. IRENE is a scanning machine used to image discs and cylinders. It's composed of high-resolution 2D/3D cameras and rotating supports for the records.



Figure 1: IRENE (Source : lab's collection)

The 2D camera takes pictures of one line on the disc, 1 pixel high for 4096 pixels long. The pictures are taken at a very high frequency while the record is turning, imaging the disc line by line. The actual frequency is tuned for the support imaged.

To take pictures, light is directed with a certain angle on the disc. The sensors are activated by the reflected light, and write the perceived amount. Direct reflection

results in white pixels, no light reflected becomes black, and in between are the gray scales.

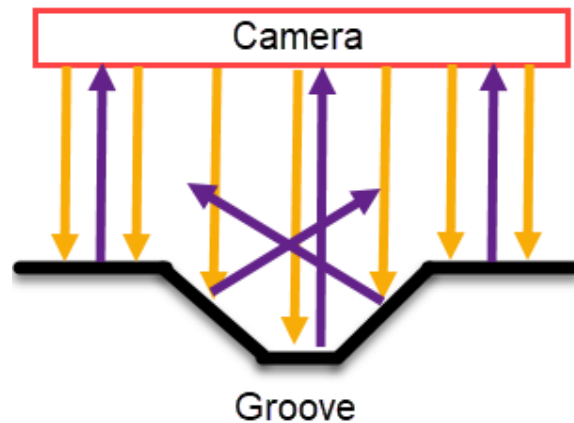


Figure 2: Camera lighting the groove and detecting direct reflection (self-made diagram)

The grooves are usually made of large white bands outside (the disc surface), black bands inside (the groove's "walls"), and a thin white in the middle (bottom of the groove). Between these bands, there is a small area of gray scaled pixels.

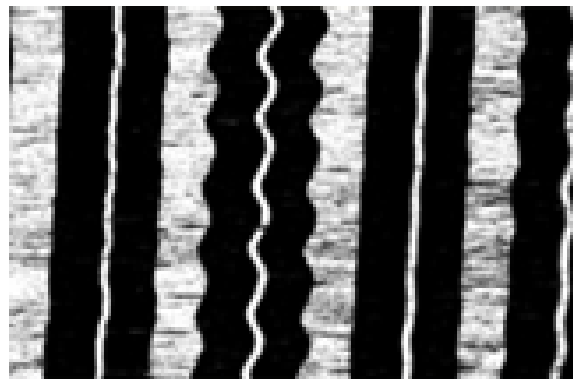


Figure 3: Resulting image of grooves (source : LBNL collection)

Any kind of damage, texture, or unwanted object on the disc can be seen, as it disturbs light reflection. It is considered as noise on the image, and the reason the ML4NR project exists.

2.3 Weaver

Along with the IRENE system exists a program named Weaver, made in C#. It is a collection of plugins developed through the years by researchers and students, used to pipeline the data reading process.

Weaver's main function is to generate the sound wave from the imaged grooves, using an edge detection algorithm to find the exact center of the groove (figure 4). However, since the images are noisy (dust, damage, texture), the sound generated also contains a high frequency background noise. Simply cutting the high frequencies wouldn't completely work, as the noise in images can also influence lower frequencies.

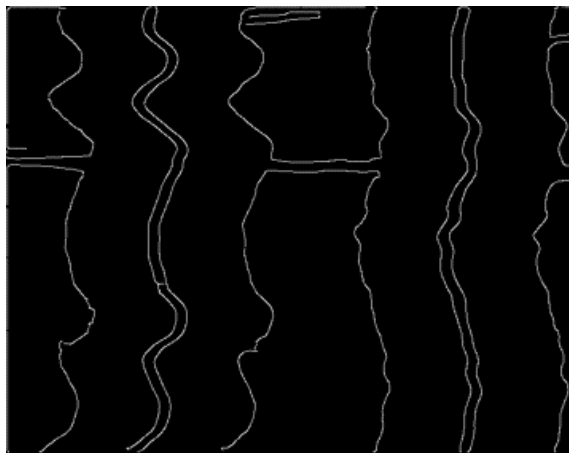


Figure 4: Detected edges. Averaging them gives the center of the groove. (Source : LBNL collection)

2.4 Machine learning and deep neural networks

Machine learning is a powerful computing technique used for handling problems having too many parameters for a deterministic algorithm, for example object detection and classification in pictures, or market predictions. It is based on the ability to learn from given correct solutions in order to predict results of future inputs.

Deep neural network is a type of model inspired by the human brain. It is based on multiple layers of neurons. Each neuron takes some data as input, does some transformation, and feeds the output to the next neuron layer. By setting a goal (ground-truth), it can evaluate its result using a loss function, and change slightly the transformations occurring in all layers (weight adjustment). It repeats this process over and over until a threshold is passed.

2.5 Shellac disc images properties

This project is aimed at shellac discs, detailed in section **Mechanical Audio Records**. The LBNL has already imaged a lot of them, and so possesses a consequent collection of groove images.

The shellac discs were imaged at 104 KHz by IRENE, resulting in exactly 80,000 pixels for one revolution. However, those revolutions are divided in 8 parts of 10,000 pixels each.

IRENE's 2D camera horizontal resolution is 4096 pixels. To obtain a good image quality, it was calibrated to take 3 microns per pixel. This means between 9 and 11 grooves are taken at the same time, and 15 to 30 revolutions are needed to image the entire disc.

So, each shellac disc is imaged with 8 x (15 to 30) pictures of 10,000 x 4096 pixels. All these pictures are in bitmap format and in gray scale.

2.6 Audio data

For some of the shellac discs, the LBNL possesses the audio source used to press them, coming from a tape. This gives us a clear reference on what the play back should sounds like. They were typically recorded at 44'100 Hz.

For the other discs, the audio reconstructed by Weaver can still serve as a good goal for the machine learning model.

3 Objectives

The main goal of the ML4NR project is to generate a cleaner sound than the deterministic algorithm when given noisy groove images as input by using machine learning. It was never attempted before at the LBNL. The project is divided into progressive steps, each corresponding to a working prototype.

3.1 Prototype 0

This prototype serves as auto-formation for using KERAS. It'll teach the basics of model building. Its goal is to have a result on the CIFAR-10 dataset (not necessarily good).

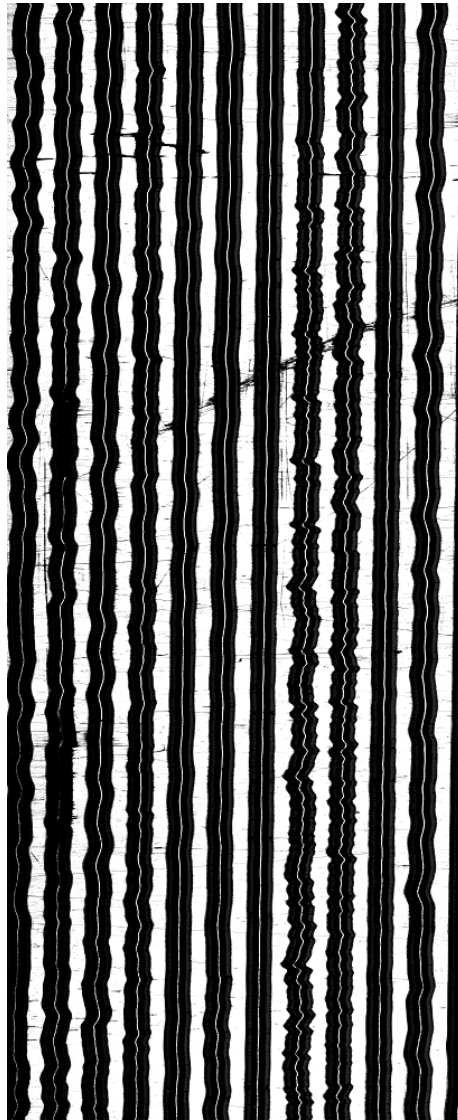


Figure 5: One image of the shellac disc of Vaya Con Dios, by Les Paul and Mary Ford (source : LBNL collection)

3.2 Prototype 1

This prototype is a proof of concept. Using Weaver, groove images of pure sine waves are generated. They are used to train a model, whose goal will be to find the simple frequency sound wave from the image. A metric needs to be defined for result evaluation.

3.3 Weaver Plugin

The previously used Weaver plugin needs to be modified, in order for it to be able to generate noisy groove images. The artificial noise needs to have the same properties as the real one from imaged grooves.

3.4 Prototype 2

This prototype tests the robustness of the model built before. It uses the noisy groove images generated with the Weaver plugin as input, and tries to find the clear sound wave. State of the art (SOTA) analysis helps improve the model.

3.5 Prototype 3

This prototype checks if complex sound waves can be generated by the model. Actual images of discs are used as input, and the goal is to find the same sound that is generated using the deterministic algorithm. Maybe a "bad result" regarding the goal could be better than current results.

3.6 Prototype 4

Final prototype, it uses the actual images of the discs, and the corresponding clean sound as goal. The clean sound is either obtained from a recorded stylus play, of from the original audio tape (which was used to print the disc). The results of this prototype are the results of the project.

If possible, this prototype is implemented as a plugin in Weaver.

4 Tasks

4.1 Prototype 0

Familiarize with KERAS and TensorFlow Read documentation and set up a working environment for KERAS. Learn how to use.

Analyze layer types Read about the types of layer commonly used when building a machine learning model. Find specific ones for this context.

Build own model Create a model that can deal with pattern recognition, for handwritten characters.

Train and test on MNIST dataset Train model on MNIST dataset (handwritten numbers) and test accuracy/precision. No need to have good results, it is a proof of concept.

4.2 Prototype 1

Analyze SOTA of sound generation Analyze the current State of the Art in sound generation using machine learning.

Define a metric for model evaluation Define which metric should be used to determine if a model is good or not.

Documentation of SOTA Produce a written document on current SOTA, for future use.

Familiarize with Weaver Learn how to use Weaver, the pipelining tool for IRENE.

Generate dataset of sine grooves Use Weaver to generate a dataset of groove images whose base is a simple sine wave (one single frequency). The groove should be clean.

Create or use model for sound generation from images Based on the SOTA, create or use a model able to generate sound from groove images.

Train and test model Train the model with the dataset previously generated. Use pure single frequency sound as ground-truth. Test with metric defined earlier. Improve the model and parameters to have better results.

Documentation of prototype Document everything about the prototype.

4.3 Weaver Plugin

Familiarize with plugin code Read and understand the code of the Weaver groove image generator plugin.

Analyze structure and properties of noise Analyze all the properties of actual discs images noise.

Implement modified plugin to generate noisy grooves Modify the plugin so it can generate noisy grooves, with a noise having the same properties as actual discs images.

Document new plugin Document everything used for this plugin, as well as noise properties.

4.4 Prototype 2

Generate dataset of noisy sine grooves Use previously implemented plugin to generate a dataset of images with grooves of simple sine waves, but noisy.

Train and test previous model Try to train previous model with noisy images, and see what it can do. Use pure single frequency sound as ground-truth (not noisy sound). Use the metrics defined.

Analyze SOTA and leads to improve model Use SOTA documentation and leads from workshops to improve the model and the results, regarding the metrics.

Document prototype Document everything about the prototype.

4.5 Prototype 3

Obtain dataset of imaged discs and generated sound Obtain dataset of actual noisy grooves, imaged by IRENE, and their corresponding sound generated with Weaver.

Train and test previous model Use model made for simple frequencies and test results. See if it can find the actual sound. The ground-truth is the noisy sound generated by Weaver.

Improve model with leads and SOTA Check documentation and leads, improve model and parameters until it obtains a good result regarding the metrics.

Document prototype Document everything about the prototype.

4.6 Prototype 4

Obtain dataset of imaged discs and recorded sound Obtain dataset of actual noisy grooves, imaged by IRENE, and their corresponding sound recorded when played with a stylus. The sound could also come from the original tape.

Train and test previous model Use model made previously and test results. See if it can find the actual sound. The ground-truth is the clean sound obtained by stylus play or from the original tape.

Improve as much as possible Check documentation and leads, improve model and parameters until it obtains a good result regarding the metrics.

Document prototype Document everything about the prototype.

4.7 Integration to Weaver

Adapt code, I/O, to integrate with Weaver If necessary, find a way to implement the model in a Weaver plugin, so it can be used for pipelining.

Document how to use Document the plugin code and its usage.

4.8 Planning

See end of document. The PDF version is in high quality and can be zoomed on.

4.9 Risks

If the sound generation from images doesn't give good results, the method can be changed. A possible method would be to "denoise" the discs images or the reconstructed sound wave, using denoiser models of machine learning.

5 Prototype 0

This prototype's goal is to familiarize ourselves with Keras and TensorFlow, by implementing a simple model.

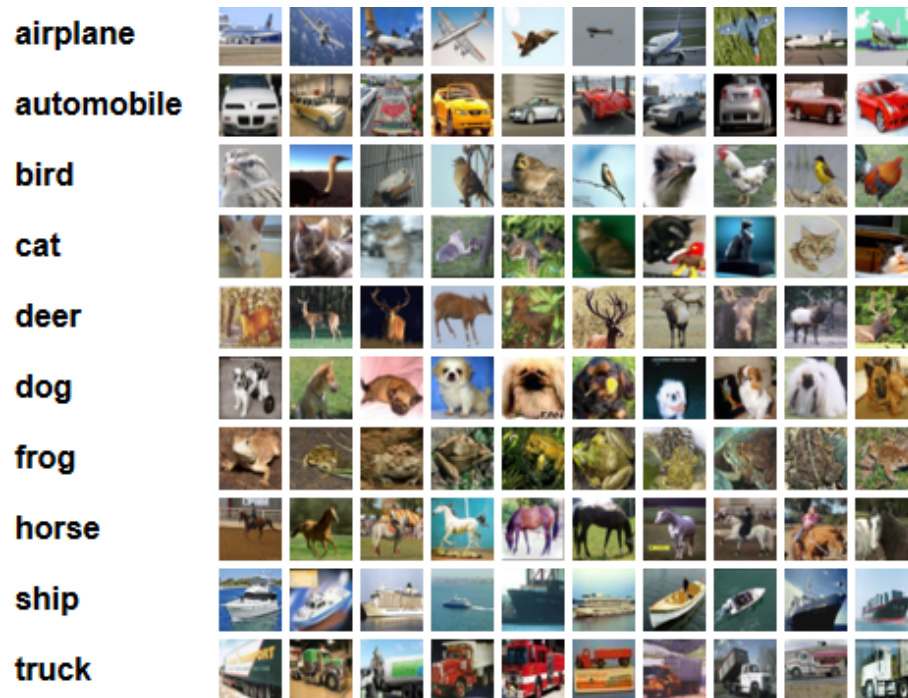


Figure 6: Example of images in the CIFAR-10 Dataset (source : [4])

5.1 Keras and TensorFlow

TensorFlow is a Python and C API developed for machine learning[2]. It facilitates computation using flow graphs, and can be used on a GPU. Interconnected neuron layers are a flow graph.

Keras is a high-level API that can use TensorFlow as a basis for machine learning and neural networks[3]. It makes it easier for developers to implement deep neural networks (DNN) by having layer types already ready to use and ways to organize them. It also offers a lot of data manipulation for pre- and post-processing.

5.2 CIFAR-10

CIFAR-10 is a public dataset widely used for machine learning training and evaluation[4]. It consists of 60,000 color images belonging to 10 classes (animals and vehicles). The goal is to be able to find the right class for a given image.

5.3 Convolutional Neural Networks

Following the tutorials proposed on Keras website[3], it seems convolutional neural networks (CNN) are a good way to start for image classification.

It is based on the principle of scaling: some features are small and very specific, while other are big and generic. CNN uses convolution to reduce the image's dimension while taking every pixel into account. The weights give more or less importance to some scales and positions in the picture.

Putting multiple layers of convolutions completely interconnected let the model finds high-level information on the picture. Usually, in between layers there is an activation function, whose task is to put values in a certain range.

Multiple iterations of 3-4 convolution layers + activation layers, followed by a max pooling layer, and ending with some fully connected layers, is the suggested way for image classification in their tutorial.

5.4 Implementation and results

The model is constructed as follow :

It obtains around 75% of accuracy for classification after only a few epochs

6 Prototype 1

This prototype will determine if it's possible to generate sound from an image. Its goal is to find the sound wave of a simple sine groove.

6.1 SOTA machine learning models

Over the years, multiple types of layers and neural networks were invented, each for a specific task[5]. In this project, the goal is to generate precisely a sound wave from a groove image. It needs to be able to find the sound amplitude at a certain point on the groove. The amplitude is given by the perpendicular velocity of the needle when following the groove, meaning it's not the absolute position that is important, but the change of horizontal position.

Machine learning models generally try to find a solution for two types of problems : classification and regression.

Classification can be done while knowing what classes are possible, or just trying to cluster inputs together. If used with images, they are usually small (28x28 to 128x128).

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 32, 32, 32)	896
activation_13 (Activation)	(None, 32, 32, 32)	0
conv2d_10 (Conv2D)	(None, 32, 32, 32)	9248
activation_14 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_5 (MaxPooling2)	(None, 16, 16, 32)	0
dropout_7 (Dropout)	(None, 16, 16, 32)	0
conv2d_11 (Conv2D)	(None, 16, 16, 64)	18496
activation_15 (Activation)	(None, 16, 16, 64)	0
conv2d_12 (Conv2D)	(None, 16, 16, 64)	36928
activation_16 (Activation)	(None, 16, 16, 64)	0
max_pooling2d_6 (MaxPooling2)	(None, 8, 8, 64)	0
dropout_8 (Dropout)	(None, 8, 8, 64)	0
flatten_3 (Flatten)	(None, 4096)	0
dense_5 (Dense)	(None, 512)	2097664
activation_17 (Activation)	(None, 512)	0
dropout_9 (Dropout)	(None, 512)	0
dense_6 (Dense)	(None, 10)	5130
activation_18 (Activation)	(None, 10)	0
Total params: 2,168,362		
Trainable params: 2,168,362		
Non-trainable params: 0		

Figure 7: Architecture of the model for prototype 0

Regression is the search of a continuous value. If the data points are time dependent, a typical problem is to predict the next value.

For classification, our problem (finding the amplitude at a given moment) could be seen as classifying small groups of groove lines into an amplitude, since they are discrete (encoded in 32 bits maximum, 4 billions classes). This seems completely unreasonable.

For regression, the groove and sound could be seen as time series, and the goal would be to predict amplitude values.

Also, the problem can be seen as an interpolation one : given clean needle positions, what is the best curve to connect them ? Once determined, taking the derivative would give the sound wave.

If it is not possible to generate sound from images, other approaches can be taken (see Risks)

Here is a list of models, their application, and their relevance to our problem:

Convolutional Neural Network - CNN Can find detailed and general information on a picture, used for classification. Outputs a vector of class probabilities. This can be used for ML4NR by treating the output vector as a list of amplitudes in playing order.

Auto-encoder Can remove noise by first finding the feature of the data while reducing it (encoding), and then expanding the feature to recreate the data (decoding). Used for image denoising, and interpolation. It can be used for this project.

Generative Adversarial Network - GAN Composed of two parts competing against each other : one part tries to generate realistic data while the other tries to detect real and generated data. Used for dataset generation when real data is hard to obtain. Can be used for denoising. May apply in this project.

Residual Network - ResNet Very deep network of convolutional layers (up to 150). The idea is to connect the input of a layer to the next layer, so they can learn by seeing processed and unprocessed data. Used for classification. Can be used here.

Recurrent Neural Network - RNN Simple neural network, but the output of a layer is fed back to a previous layer. This allows it to learn from past values, and is good for time series. Used for regression. Can be used here.

Long Short-Term Memory - LSTM Same idea as recurrent neural networks, it uses previous feature values to find the present output. The difference is the

possibility to have as many previous data as needed. A special neuron is responsible for telling when a previous feature value is still relevant or not. Very good for time series, can be applied here.

By tweaking the problem, all those models could be used. It is complicated to see what would be best, as no similar project were found. For this prototype, the first try is with a CNN model, like the one used in prototype 0. The second try is with a LSTM model.

6.2 Risks

If the model is really bad at finding the correct sound from the image, other methods could be used :

- Cleaning the image of its noise with an auto-encoder.
- Finding the needle positions at successive time steps, and using the derivative to obtain the amplitude
- Cleaning the generated audio of its noise

6.3 Weaver plugins

Weaver consists of more than 150 plugins, each doing a specific process. They are divided into categories : Load, Image, Track, Process, Save, Other.

All plugins inherit from the AbstractPlugin class, containing data useful to most of plugins. They are chained automatically when placed in the pipeline.

6.4 Dataset generation plugin

A machine learning dataset needs to be as diverse as possible, in order to avoid over-fitting. A plugin able to generate a random dataset would be ideal.

A single Weaver plugin, named Create2D_Sine, manages the generation of clean simple groove images. Different parameters can be chosen. A new plugin inspired by Create2D_Sine is implemented in order to generate an entire dataset in one go, with the following ideas in mind:

- The images should be random in an acceptable range.
- It should be possible to generate systematically all combinations if desired.

Width	204
Height	80'000
Frequency	20 - 5'000
Amplitude	10
Edge width	3
Groove width	160
Bottom width	12

- The dataset size can become quite big, an estimation of the size would be useful.

All sine groove parameters can be randomized. The most intuitive way to do it is to take minimum, maximum and step for each parameter. Setting minimum = maximum will generate only this value for this parameter.

Figure 8 shows the interface of the newly developed plugin.

The generated groove file name contains the values used for each parameter. Then are encoded as the parameter's first letter and the value. Ex : f1875_a10_-e3_b12_g160_h80000.tif is a groove of frequency 1875 Hz, amplitude 10, edge width 3, bottom width 12, groove width 160, and 80'000 pixels of height.

Using Weaver loop plugin, the corresponding sound of each image is generated.

6.5 Dataset

Figure 9 shows what a generated groove looks like. The first idea is to generate randomized grooves so model inputs are as diverse as possible. However, the audio file generated from a random groove can sometimes be wrong. The algorithm used by Weaver plugins needs initial values close to the actual ones, and it's quite complicated with a random dataset. Furthermore, even the "correct" audio files contain some high frequency noise, due to the algorithm imperfection. We decide to generate the audio data from the code and not Weaver for this prototype.

The groove and image properties are fixed, for an easier manipulation of data for training. Only the frequency is randomized.

In order to keep some space, we generate around 100 images.

So ideally, the input is a 204 x 80'000 8-bits image, and the output is a 104KHz 16-bits/channel stereo wave file.

1 Create2DSineImag GC ☐ Display? From

2 (sec) 1

Generation

☐ Systematic ☒ Random Qty 5

Estimated size on disk : 126.266 MB

No folder selected

Frequency

Min 20.00 Step 1.00

Max 20000.00

Amplitude

Min 10.00 Step 1.00

Max 50.00

Edge width

Min 2 Step 1

Max 10

Groove width

Min 150 Step 10

Max 200

Bottom width

Min 10 Step 1

Max 15

Image height

Min 80000 Step 5000

Max 80000

Figure 8: Dataset generation plugin interface

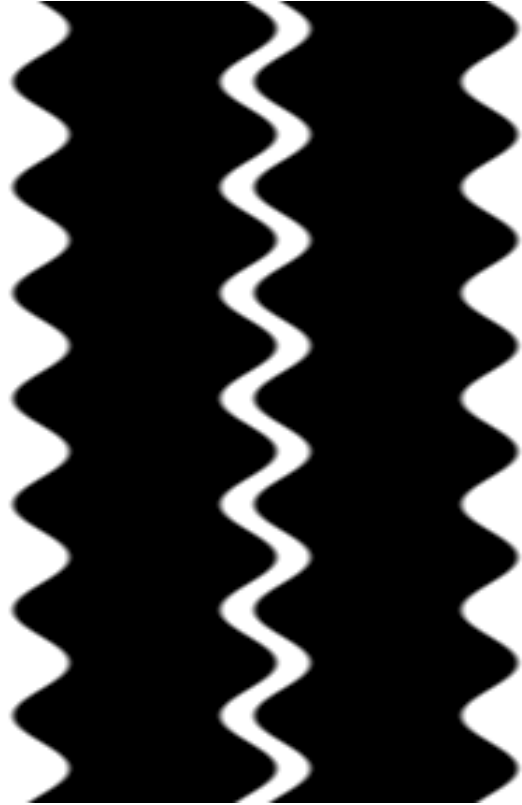


Figure 9: Generated groove, frequency=2760

6.6 Model implementation

6.6.1 Version 1

The first intuition is to use what we already know from prototype 0: the CNN layer.

The input is a 80'000 by 204 8-bits gray-scale groove image, and the goal is to find the WAV data of 80'000 samplings. The audio file has two channels, even if most of the time they have the same value. Each channel is encoded in 16-bits. We only search for one channel and copy it for the second.

The first layer has to accept one groove image as input, as they are too heavy to all be loaded in memory. For each epoch, we loop through all images, feeding them one at a time to the model for fitting. The model is saved every epoch, because Google Colab, used for the implementations of this project, makes sessions expire after a fixed time.

The convolutions kernel are 10x10 because 10 rows of data should help find the amplitude at a certain point, and 10 columns cover the groove edge. The output size of 32 is what is typically used in examples. Padding keeps the image at the same size.

A good optimizer is ADAM, who uses gradient descent. All parameters are left with default values. Mean square error (MSE) seems to be a perfect loss function for our problem, as the further we are from the ground-truth, the worse it is. Mean absolute error (MAE) can also provide insight on the results.

6.6.2 Version 1 results

After training for 10 epochs over 94 images, the best results are obtained by images whose audio contains a lot of 0. Their loss function score is however still horrible : in the order of 10'000s to 100'000s for MSE, 1000s for MAE. Images whose sounds have a lot of non-zero score in the order of 10'000'000s for MSE, and 10'000s for MAE.

Seeing that the loss scores increases over time, we know there is a problem with our parameters and/or inputs.

Since it is a first shot at this problem through intuition, we decided to abandon this method and to search a better model.

6.6.3 Version 2

The first version being unsuccessful, we searched further in the current SOTA for regression problem linked to time series. A neural network that stood out was Long Short-Term Memory (LSTM)[6].

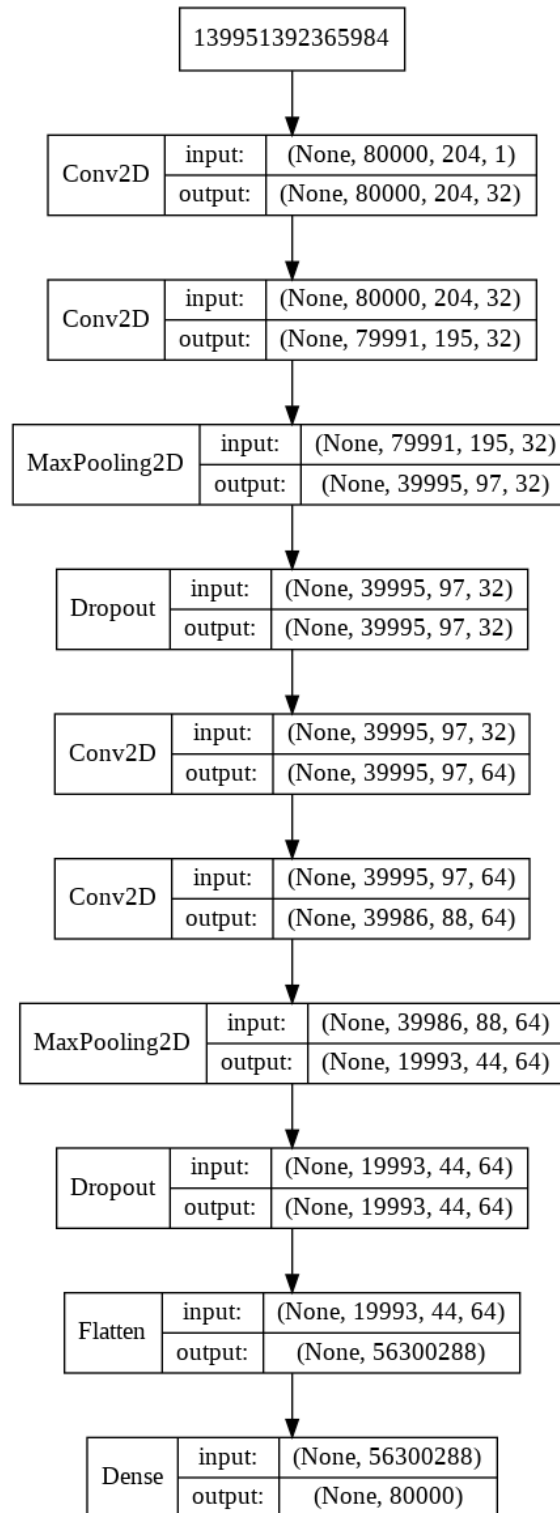


Figure 10: Architecture of the convolutional network

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 80000, 204, 32)	3232
conv2d_14 (Conv2D)	(None, 79991, 195, 32)	102432
max_pooling2d_7 (MaxPooling2D)	(None, 39995, 97, 32)	0
dropout_7 (Dropout)	(None, 39995, 97, 32)	0
conv2d_15 (Conv2D)	(None, 39995, 97, 64)	204864
conv2d_16 (Conv2D)	(None, 39986, 88, 64)	409664
max_pooling2d_8 (MaxPooling2D)	(None, 19993, 44, 64)	0
dropout_8 (Dropout)	(None, 19993, 44, 64)	0
flatten_4 (Flatten)	(None, 56300288)	0
dense_4 (Dense)	(None, 80000)	4504023120
Total params: 4,504,023,840,192		
Trainable params: 4,504,023,840,192		
Non-trainable params: 0		

Figure 11: Parameters count of the model

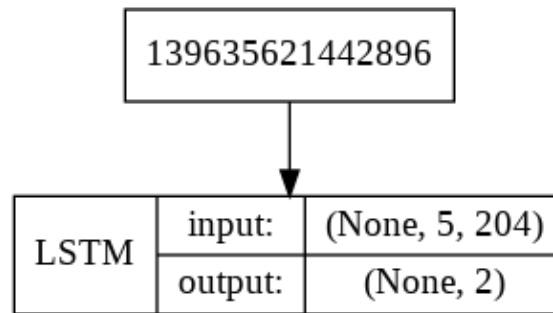


Figure 12: Architecture of the LSTM network

Layer (type)	Output Shape	Param #
lstm_4 (LSTM)	(None, 2)	1656
Total params: 1,656		
Trainable params: 1,656		
Non-trainable params: 0		

Figure 13: Parameters count of the model

LSTM are composed of neurons able to retain information from previous steps. They use them to predict values, but if an input from a previous step is no longer useful, it's able to forget it.

In order to use it with Keras, the inputs needs to be reshaped in three dimensions. The first is blocks of multiple rows. The second is the rows of features for one prediction, also called time steps. The third is the features.

Since it already is a full Neural Network (it generates a node for each feature at each time step), we decided to use only one of them for now. It will take one block of rows as input at a time, and output the amplitude for both channel at that time step.

This model takes a 5 x 204 image as input, and gives 1 x 2 as output (the amplitude for both channels).

For the loss function and optimizer, we will keep the same as version 1.

6.6.4 Version 2 - results

After training for 10 epochs over 94 images, the results are in the same range as version 1. It seems the problem comes from the learning rate and the features, rather than the model itself.

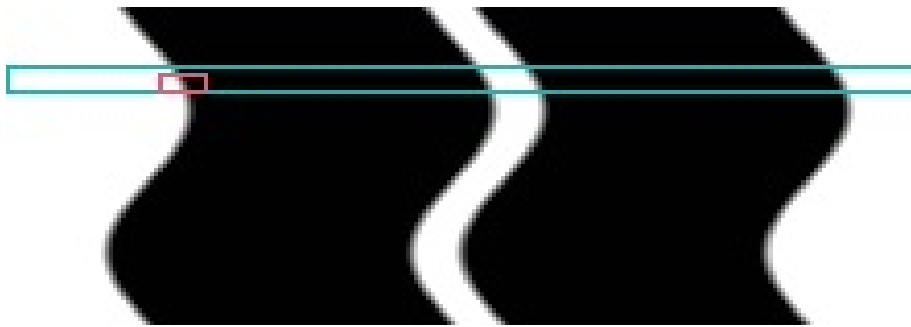


Figure 14: Window used to predict the amplitude

6.6.5 Version 3

With version 2 results, we now know that the features are bad when used directly.

Following the advice of Mr. Hennebert, we normalize the inputs and target to obtain a nicer range. Each pixel of the groove image is normalized between 0 and 1. Each audio amplitude goal is normalized between -1 and 1.

Furthermore we use the CNN again, as it is easier to understand and fine tune. However, the input will have the same shape as in version 2 : a few rows of all pixels as input, and the amplitude as output. Since the sound should be in mono, only one output value is needed.

The blue rectangle is an example of input given to the model. The red rectangle is an example of convolutional kernel.

Each image is split into overlapping blocks of 9 rows and 204 features. They are then fed to the model, in which each convolutional layer tries to find hidden features in 5x10 matrices.

For the two previous versions there were no activation layer, because the output wasn't bound. Here, since we are searching for an amplitude between -1 and 1, the tanh activation function seems reasonable.

The dropout layers make it harder to overfit.

The optimizer is also different : we use RMSprop, whose key advantage is the automatic adaptation of its learning rate. This avoids loss function triggering long jumps and divergence. The parameters used are `learning_rate=1e-4`, `decay=1e-6`.

For validation, 15% of the blocks of one image are taken out of training, and then used for scoring. Furthermore, after 25 images, a real sound generation is made from unseen images and saved (no scoring). This allows us to hear the results rather than comparing numbers.

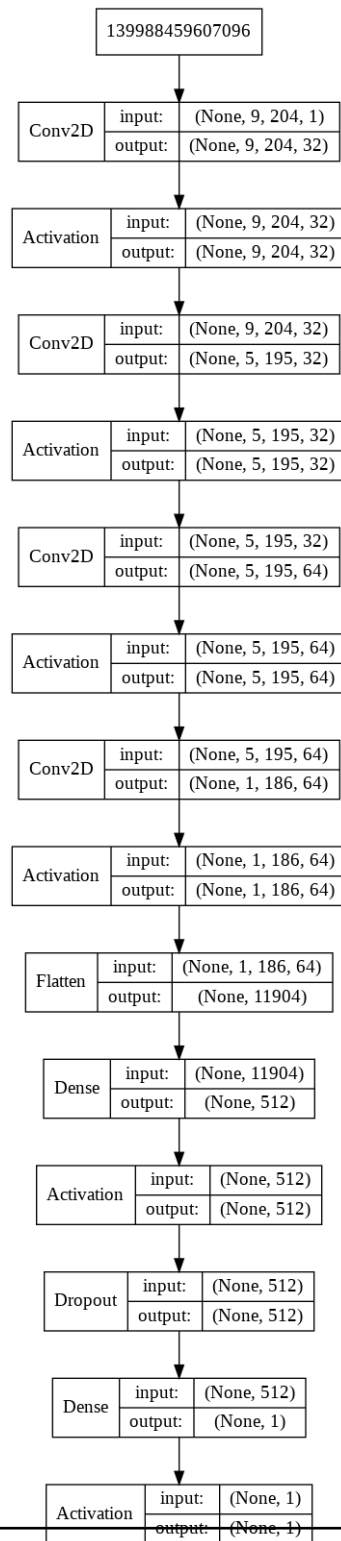


Figure 15: Architecture of the CNN with window

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 9, 204, 32)	1632
activation_10 (Activation)	(None, 9, 204, 32)	0
conv2d_14 (Conv2D)	(None, 5, 195, 32)	51232
activation_11 (Activation)	(None, 5, 195, 32)	0
conv2d_15 (Conv2D)	(None, 5, 195, 64)	102464
activation_12 (Activation)	(None, 5, 195, 64)	0
conv2d_16 (Conv2D)	(None, 1, 186, 64)	204864
activation_13 (Activation)	(None, 1, 186, 64)	0
flatten_1 (Flatten)	(None, 11904)	0
dense_1 (Dense)	(None, 512)	6095360
activation_14 (Activation)	(None, 512)	0
dropout_3 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 1)	513
activation_15 (Activation)	(None, 1)	0
Total params: 6,456,065		
Trainable params: 6,456,065		
Non-trainable params: 0		

Figure 16: Parameters count of the model

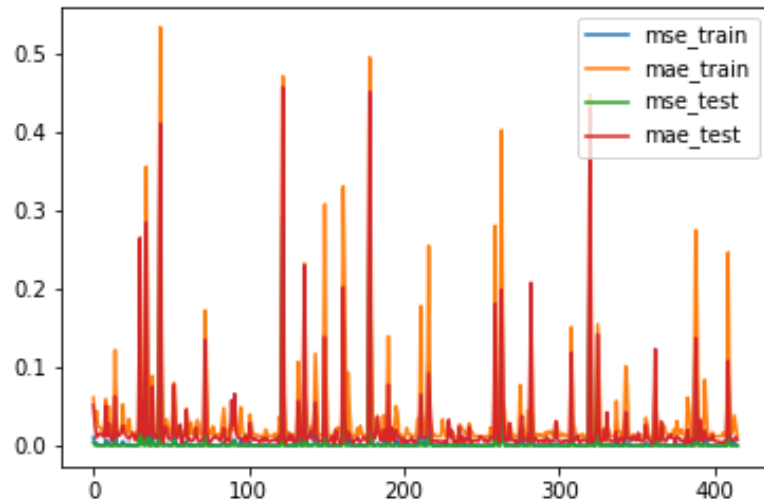


Figure 17: MSE and MAE for train and test scores

6.6.6 Version 3 - results

This model seemed to have good results from the beginning, so everything was put in place to record the training : MSE and MAE scores for train and test images, weights saving, audio file registering.

Here are the MSE and MAE scores :

We can see spikes. Those corresponds to low frequency grooves, where the lateral movement is really slow. We can see why the model has a hard time with them.

A prediction of a low frequency groove has less amplitude, but the frequency itself seems to be correct.

Expanding the range after the generation could make it better.

As the frequency increases, the prediction is more spot on, and the audio sounds closer to the truth.

It is however very sharp, and analyzing the spectrum shows that a lot of higher frequencies are involved (mostly harmonics of our desired frequency).

6.6.7 Version 4

In order to improve our results, fine-tuning all hyper-parameters is necessary. The success or failure of neural networks depends on it, as a bad combination could make



Figure 18: Groove of frequency 25

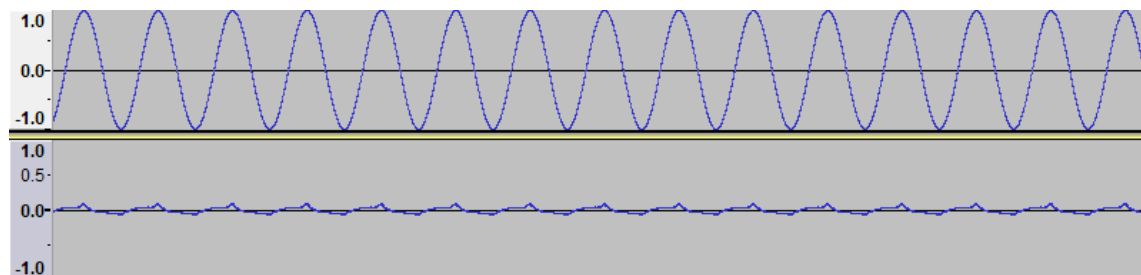


Figure 19: Top : perfect sine wave of 60Hz / Bottom : predicted wave from groove

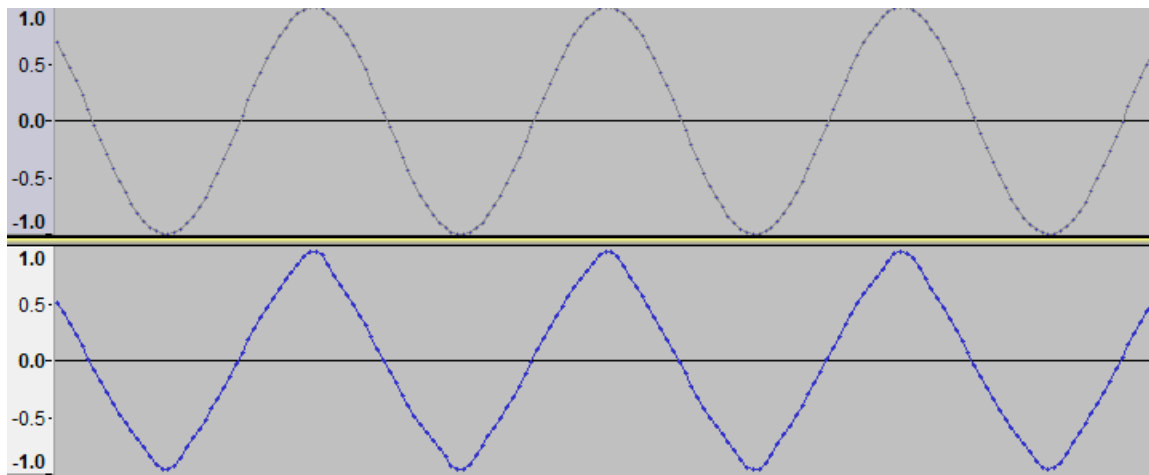


Figure 20: Top : perfect sine wave of 2175Hz / Bottom : predicted wave from groove

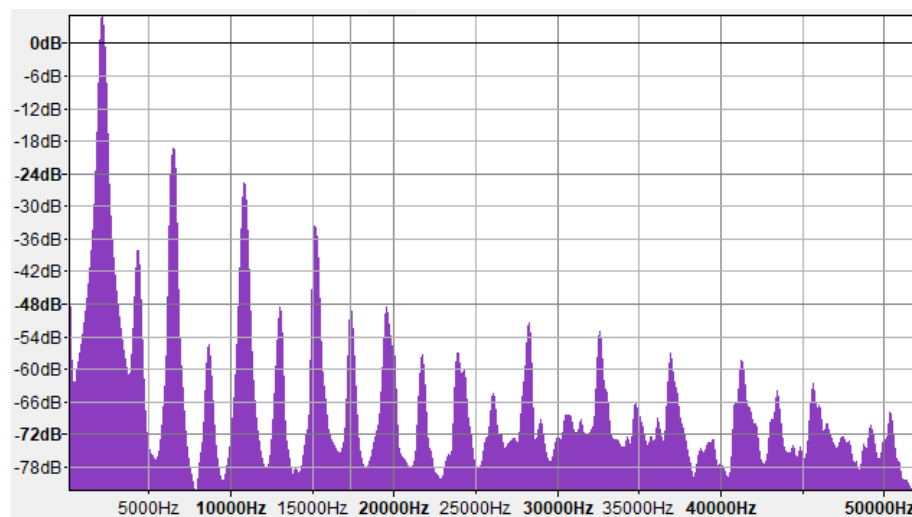


Figure 21: Spectrum of the generated audio for a 2175Hz groove

the model and results diverge instead of converging.

The hyper-parameters of our model are :

Batch size Number of blocks given to train the network before updating the weights.

Mainly depends on the memory available

Time steps Number of rows of the sliding window. Too few and it will be hard to differentiate slopes of the groove. Too many and the model will take a long time to find what is useful.

Predictions Number of amplitude points to find in one block. There may be a link between following amplitudes that the model can use to be more accurate.

Convolution filter shape Convolution kernel size. Small filters can find smaller features, big ones can detect more general elements. Generally 3x3 in CNN.

Max pooling filter shape Size of the kernel in which we keep only the maximum. Generally 2x2 in CNN.

Dropout rate Proportion of neuron connections randomly cut to avoid over-fitting and accelerate learning time.

Activation function Function that will collapse values on a certain range. **tanh** gives values between -1 and 1, **ReLU** between 0 and infinity.

Optimizer Algorithm used to update the network weights. There exists a lot of them, with different properties and applications. Most of them need a learning rate and a decay rate.

Learning rate Strength of the weight updates when searching the minimum loss. Too high and the values could diverge, too low and the learning will take a long time to find the minimum.

Decay rate Reduction of learning rate at every step, so it is less likely to diverge.

Loss function Value calculated between predicted value and goal. This value is what the optimizer tries to minimize.

This makes a lot of parameters, and testing all possible combinations is impossible. Some of them seem to be more linked than others, and using intuition, are grouped like that :

1. Time steps, predictions, and convolutional filter shape

2. Max pooling shape and dropout
3. Optimizer, learning rate, and decay rate

Batch size can be fixed, as it should not have too much of an influence (32 is generally used). Loss function can also be fixed. For regression problem, a mean square error seems perfect.

Since the dataset is not noisy like the actual grooves, some of the parameters found here may not be ideal later. For this reason, the models are not trained for too long. One pass over all images should be enough to find the good combinations.

The input dataset is 78 images, each of them decomposed in up to 79'996 blocks using a sliding window. 15% of them are taken away for validation, resulting in up to more than 5'300'000 blocks for training one epoch. The model trains on each block individually to find a certain number of predictions per block. This dataset should be big enough to have interesting results after one pass only.

In table 1, values considered for the first group of hyper-parameters can be found.

Hyper-parameter	Min	Max	Step
Time steps	5	9	2
Predictions	1	Time steps	2
Convolution filter height	3	Time steps / 2 + 0.5	2
Convolution filter width	3	9	2

Table 1: Range of values for hyper-parameters fine tuning

This already results in 104 possible combinations. In order not to lose too much time, only the 50 first combinations will be trained. If the best results are obtained with parameters values close to those that aren't trained yet, the remaining combinations will also be trained. However, if the best models are the ones with small parameter values, there is probably no need to train models with bigger values.

The other hyper-parameters seem more dependent on whether the dataset is noised or not, so they will be fine-tuned in prototype 2.

6.6.8 Version 4 results

For result evaluation, every model tries to predict the audio wave of the 18 test images. The mean square errors of each prediction are averaged together to obtain the global test result.

The best results, after training over 78 images of grooves with variable frequency, are obtained by the first model tested (results of best three models in figure 22).

```
Best : model1 at epoch 1 with score 0.006936517560244762

Second : model4 at epoch 1 with score 0.008222158477872332

Third : model3 at epoch 1 with score 0.00875592185037822
```

Figure 22: Top three results obtained by different combinations

Time steps	5
Predictions	1
Convolution filter	3x3
Max pooling filter	2x2
Dropout rate	0.25 (firsts) and 0.5 (last)
Activation function	ReLu (firsts) and tanh (last)
Optimizer	RMSprop
Learning rate	1e-4
Decay rate	1e-6

Table 2: Hyper-parameters used for best results

All three models have small hyper-parameters values : all have time steps = 5, predictions = 1, and convolution filter height = 3. The only difference between them is convolution filter width, which is in order 3, 7 and 9.

All the other results can be found in the appendices.

It can be assumed that there is no need to train models with bigger parameter values.

In table 2, you can find the values used by the best model.

The model architecture can be found in the appendices.

For reference, the goal and predicted signals can be seen in figure 23. Compared to the previous prediction of the same groove (figure 20), this one looks better, especially at the peaks.

The spectrum (figure 24) also shows a big difference between the main frequency and the overtones, indicating a reduced background noise

7 Prototype 2

In this prototype, the goal is to train a model on noisy images of single frequency grooves and make it predict a perfect sine wave.

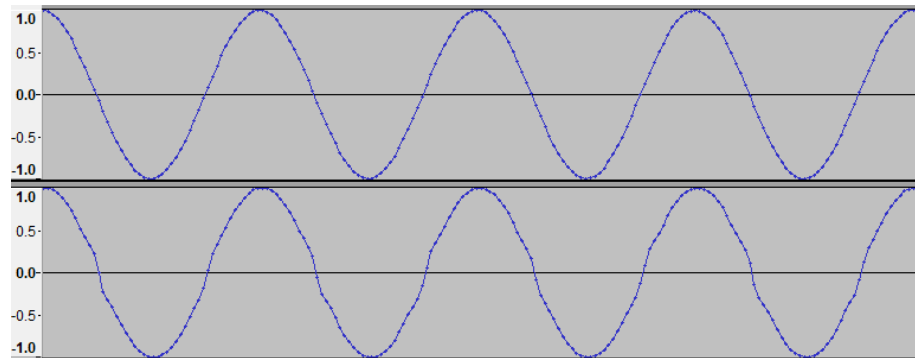


Figure 23: Top : goal signal of 2175Hz. Bottom : predicted signal

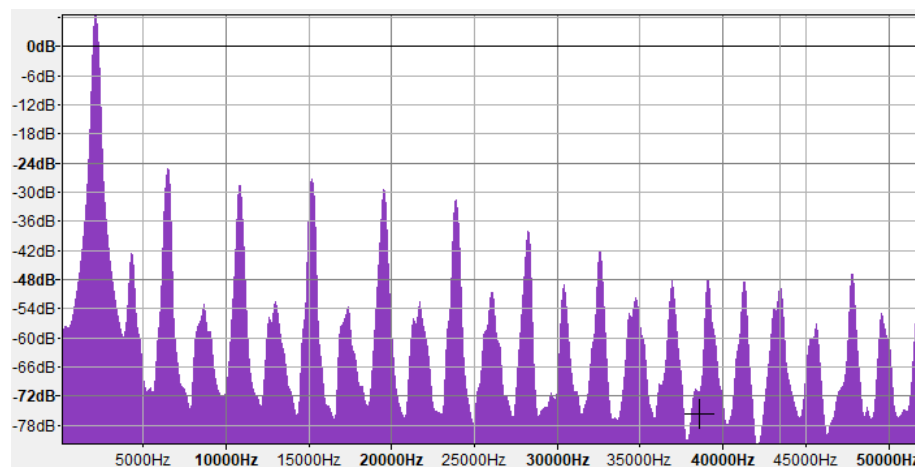


Figure 24: Spectrum of the predicted audio wave

7.1 Noisy groove images

As can be seen in figure 25, the actual disc images are noisy. That noise can be decomposed in different type of noise :

- A light Gaussian noise that gets heavier on the groove edges, caused by dust and texture.
- Thin scratches that are seen as black.
- Bigger damage that is also seen as black.

All those makes the task of finding the sound more difficult, it is therefore important to simulate them for model training.

7.2 Artificially noised dataset

Since a dataset of grooves and their output is already available, a simple program can be used to noise them and generate a new dataset. Those noised grooves should look as much as possible as the actual disc images.

The Gaussian noise can be added easily on the perfect groove image. The black part should stay as dark as possible, and the white part should have visible deterioration. Knowing that the pixel values are between 0 and 255, a Gaussian noise of average -100 and standard deviation of 50 is clearly visible on the white parts and not so much on the black ones.

For the scratches, thin long black rectangles are generated randomly and placed randomly across the image.

For the big damage, the same is done with bigger black rectangles, but not as many as scratches.

The result of artificial noising can be seen in figure 26.

7.3 Version 1

In this version, the dataset is the same as for prototype 1 (78 images for training, 18 for testing). All the images are noised once and written in static memory, so the training and testing images are the same for all models.

The goal is to obtain a good prediction of the sound wave of a noisy groove image. In order to do that, further fine tuning is necessary. Instead of trying every combination like before, parameter values are tuned through trial and error.

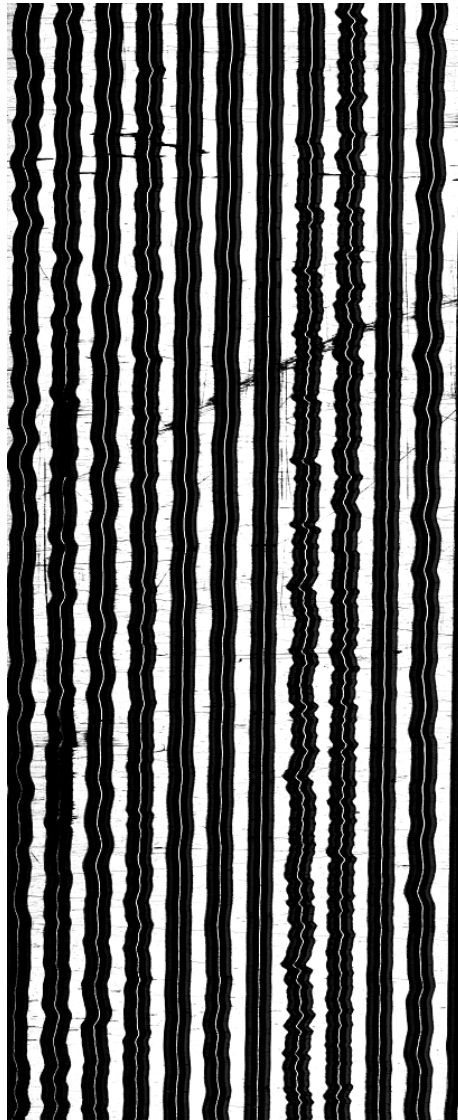


Figure 25: One image of the shellac disc of Vaya Con Dios, by Les Paul and Mary Ford (source : lab's collection)

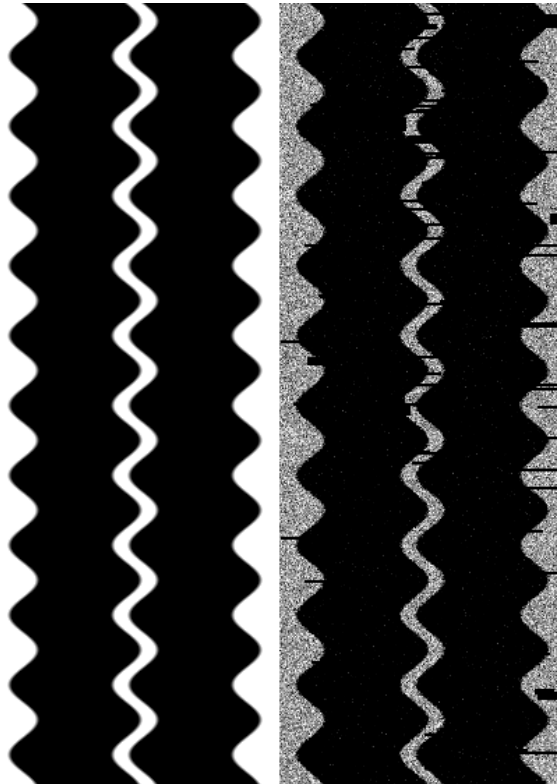


Figure 26: Left : clean groove of 2075 Hz. Right : same groove artificially noised

```
Best : model3000 at epoch 5 with score 0.03106682899938857  
Second : model3000 at epoch 11 with score 0.031149510099533793  
Third : model3000 at epoch 4 with score 0.0316330659933328
```

Figure 27: Top three results among all epochs and models where learning and decay rates are tuned

7.3.1 Implementation

The first model trained is the one that obtained the best results in prototype 1.

Then, fine tuning over the learning and decay rate are tried. In the first model RMSprop is used, and the values are learning = $1e-4$, decay = $1e-6$. However, Keras documentation suggest using default values, which are learning = $1e-3$ and decay = 0. We can also try to make the decay bigger to see how it affects learning.

Seeing the noised images, it seems a bigger sliding window could help find amplitude more easily when the groove edge is damaged. Big jumps in values make it faster to see if it's relevant or not to change it, so the network is tested with window height (time steps) of 5, 9, and 15.

Each model is trained over a varying number of epochs, depending on the occasional crash of the connection and the time taken.

7.3.2 Results

First come the results between the different learning and decay rates. Figure 27 shows the results of the three best model and epoch.

The model using learning rate = $1e-4$ and decay rate = $1e-6$ obtains the three best results. However, the results don't seem to improve over time. It can be a sign of bad values or even bad optimizer.

Next, using the default values of the RMSprop optimizer, the window height parameter is played with. Again, figure 28 shows the results of the three best model and epoch.

This time, the results are a bit better. The best model is the one using a window height of 15 pixels. Our intuition proved to be right.

```
Best : model3003 at epoch 4 with score 0.02539694647420396  
Second : model3003 at epoch 7 with score 0.02618928231902284  
Third : model3003 at epoch 3 with score 0.02693568985958112
```

Figure 28: Top three results among all epochs and models where sliding window size is tuned

7.4 Version 2

In this version, a new dataset is used : groove images are now lightly randomized (groove width, edge width, bottom width), and they are noised on the fly during training. The test dataset is noised once, written in memory, and used for every model so comparisons make sense.

The training dataset contains 170 images, the test 30 images. 15% of image blocks are used for validation. This makes training for one epoch quite long (around 3 hours with our current environment). Furthermore, a good result over the test set could be obtained in the middle of an epoch. With these two arguments in mind, a modification was made to the program : it is now possible to divide the training dataset and to test results after each smaller division. This also allows to resume training in case of crash.

In the previous version, the optimizer didn't seem to work correctly even with different values. In this version, other optimizers are tested : Adagrad and Adam.

7.4.1 Implementation

First, the best hyper-parameters previously found are used to train on the dynamic dataset. The same data is never seen twice, so the model should be more robust.

Then, optimizers Adagrad and Adam are tested with their default values.

The training dataset is divided in 10 smaller parts, and after each of them the model is tested.

7.4.2 Results

The model combining the best hyper-parameters and RMSprop optimizer trained over 6 epochs. The best results are obtained on epoch 1, showing that there is indeed a problem with the optimizer. However, it was used to make a prediction and see if the model can find the audio of a noised groove.

Figure 29 is the noisy groove used for test (never seen during training). Figure 30 is the perfect wave and the predicted one. Figure 31 is the spectrum of the predicted audio wave. The predicted wave looks thicker on the peaks. We can see that there are still overtones, however they quickly lose intensity. The predicted wave has some irregularities, but they are really small compared to the rest. It seems that even though it doesn't converge, this model is robust enough to predict a sound wave from a noisy image.

The next model to test has the same hyper-parameters except for the optimizer. It uses Adam with its default rates. The best test results are obtained at epoch 2.6. This epoch weights are used to make a prediction on the same groove as before. In figure 32 we can see the predicted sound wave on third position, the second being the prediction using RMSprop optimizer. We can see that the general shape is more similar to the perfect wave, however there are sharper errors. The spectrum (figure 33) also shows overtones with less amplitude. It is a better result altogether.

8 Prototype 3 and 4

The goal of this prototype is to obtain a clean sound wave from real disc images. There are three alternatives to try and find the audio from disc images :

- Giving disc images directly to a previous model, and see what it can find.
- Continue the training of a previous model but with actual disc images as input and clean tape audio as goal (transfer learning).
- Train a new model only on disc images and tape audio.

Prototype 3 as described in the project objectives is abandoned. Indeed, training a model on noisy disc images with a noisy sound wave as goal seems it would only make it learn to include said noise in his predictions.

8.1 Dataset

Over the years, multiple discs have been imaged by researchers at the LBNL. Among those, 4 discs are available with their original tape audio. These seem perfect for the project, however they need some manipulation to be usable.

Each disc is digitized in multiple 80'000 by 4096 images. One image represents a complete revolution of the disc, but only covers around 10 grooves in width. 15 to 30 of those images are needed to represent the entire disc. Because the camera is placed

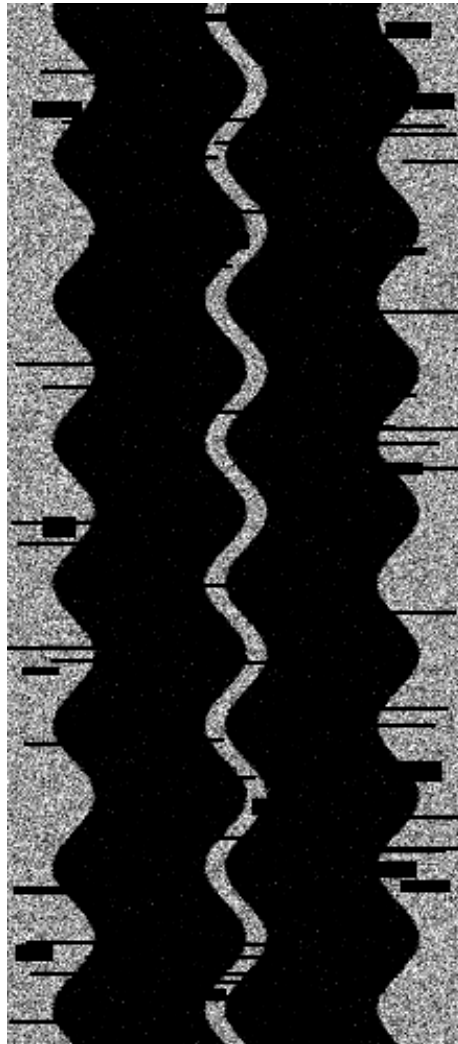


Figure 29: Noisy 1530 Hz groove image used for testing

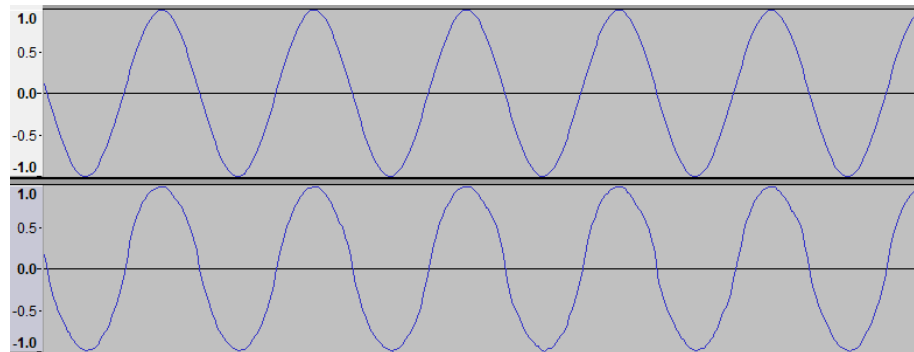


Figure 30: Top : perfect 1530 Hz wave. Bottom : predicted wave from noisy groove image

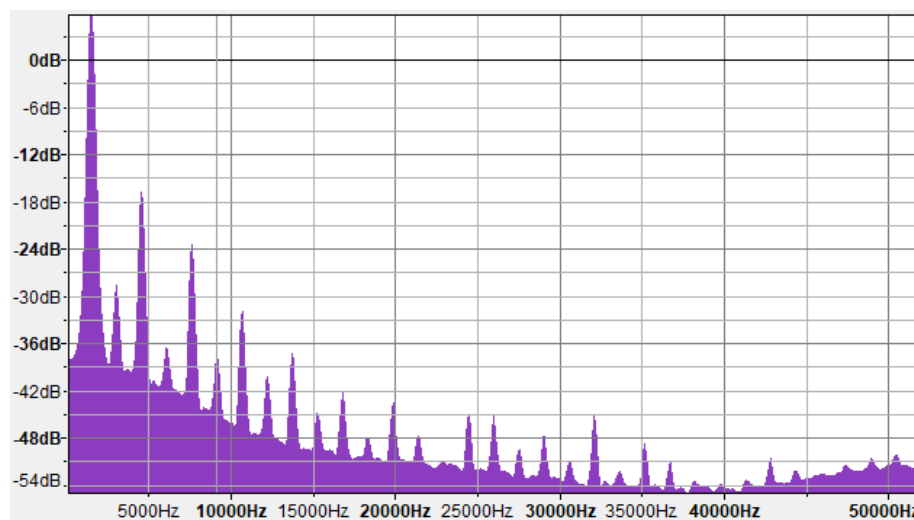


Figure 31: Spectrum of the predicted 1530 Hz wave

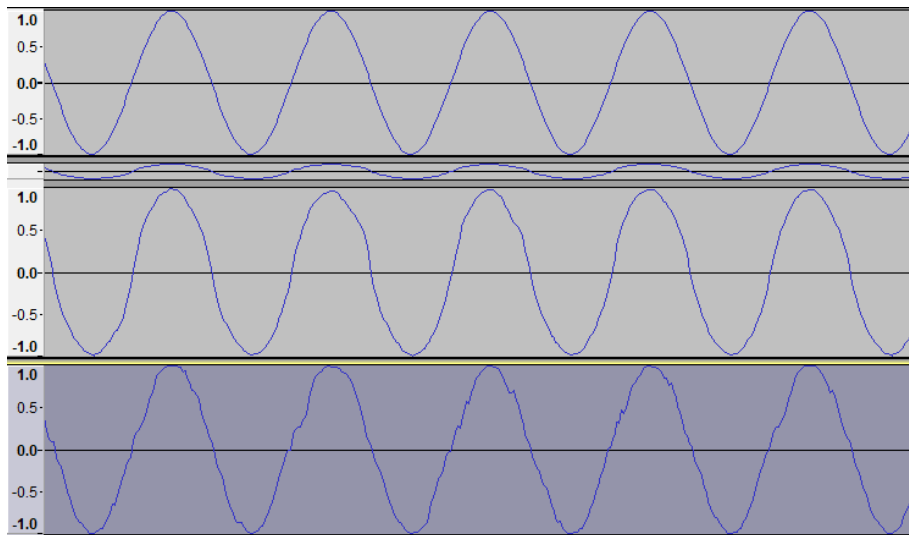


Figure 32: Top : perfect 1530 Hz wave. Middle : predicted wave from noisy groove image with RMSprop. Bottom : predicted wave from noisy groove image with Adam

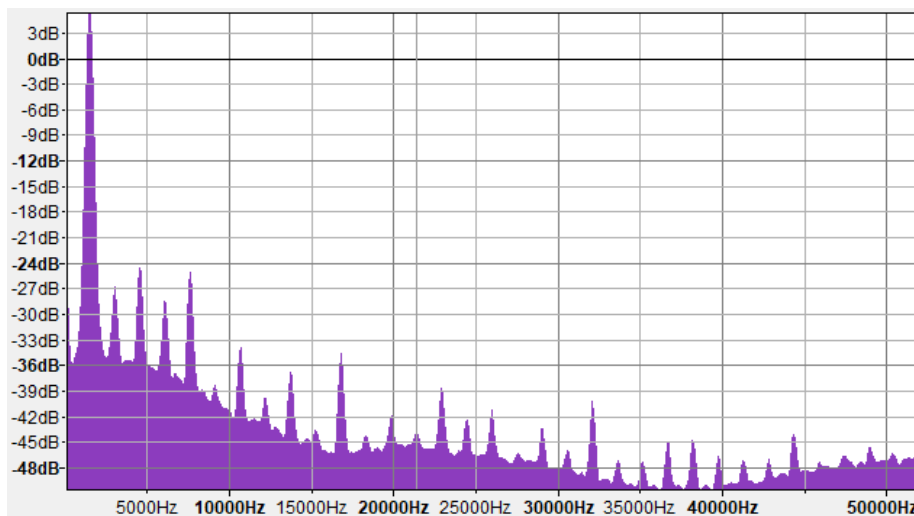


Figure 33: Spectrum of the predicted audio wave

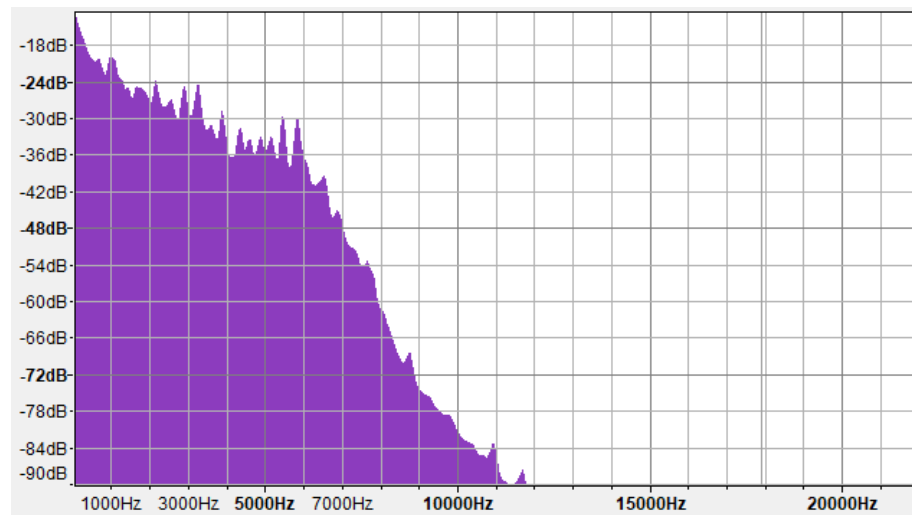


Figure 34: Tape audio spectrum of Johnny is the boy for me by Les Paul and Mary Ford (generated in Audacity[?] with Hanning window)

and calibrated manually, the images overlap. Furthermore, each image is divided in 8 parts of 10'000 by 4096 pixels for storage. Those part don't overlap, and a Weaver plugin can merge them to obtain the bigger images.

For each of the 80'000 by 4096 images, multiple audio files have been generated using Weaver and its edge detection algorithm, trying different combinations.

The audio coming from the original tape is the audio of the entire disc. It is very clean, as no background noise can be heard. Figure 34 shows the spectrum of one of those tape audio : Johnny is the boy for me by Les Paul and Mary Ford.

This audio is sampled at 44'100 Hz, but since IRENE has an imaging frequency of 104'000 Hz for shellac discs, it would be better to re-sample to also be at 104'000 Hz instead.

We need the disc images to match with the tape audio. There are two reasonable workarounds to do that :

- Divide the tape audio in parts corresponding to each 80'000 by 4096 image.
- Merge all disc images, searching for overlapping parts.

It is worth noting that each 80'000 by 4096 image takes around 300 MB of space. This means that an entire imaged disc takes between 4.5 and 9 GB of space. While this is reasonable for non-volatile memory, that space is needed in RAM when actu-

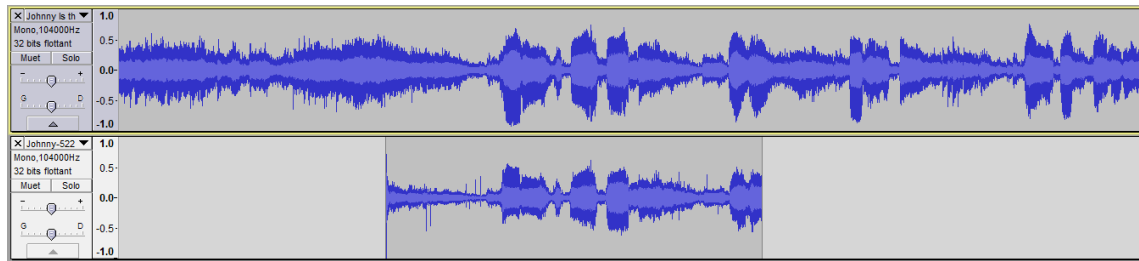


Figure 35: Trying to align the generated audio on the clean tape audio. Top : tape audio. Bottom : Weaver generated audio

ally merging the images, and our computer is not powerful enough for this. As so, the first workaround seems to be the best choice.

Since there are no indication on where the audio starts on the disc, we need to search manually for the right part. Weaver can help us for that : using the edge detection algorithm, the audio of one 80'000 by 4096 image is generated. Then, this audio is compared and aligned with the tape audio by eye and ear in an audio editing software (Audacity[?] was used for this project). It can be seen in figure 35.

The rest of the tape audio is cut out. The clean audio corresponding to one image is now available.

The last problem comes from the images themselves. Since they cover multiple grooves (it is one groove, connecting at the top and bottom), we need a way to isolate them. It is not possible to divide the image in straight bands containing one groove each, because of a common defect in discs : the middle hole is not perfectly centered. It provokes a global shift in groove position. Figure 36 is a vertically compressed 80'000 by 4096 image showing that shift.

Again, using Weaver's edge detection algorithm can help. It is used to find the middle of the grooves at each line and for each of them. It is imperfect because of the image's noise, but it is enough for what we want : each point of the newly found track is used as the center of a sliding window. In figure 37, the plugins used for edge detection and their parameters can be seen. Figure 38 shows the track actually found.

All parts are now ready for machine learning : groove image used for input, the track for the sliding window, and the clean audio goal, with one amplitude for each track position.

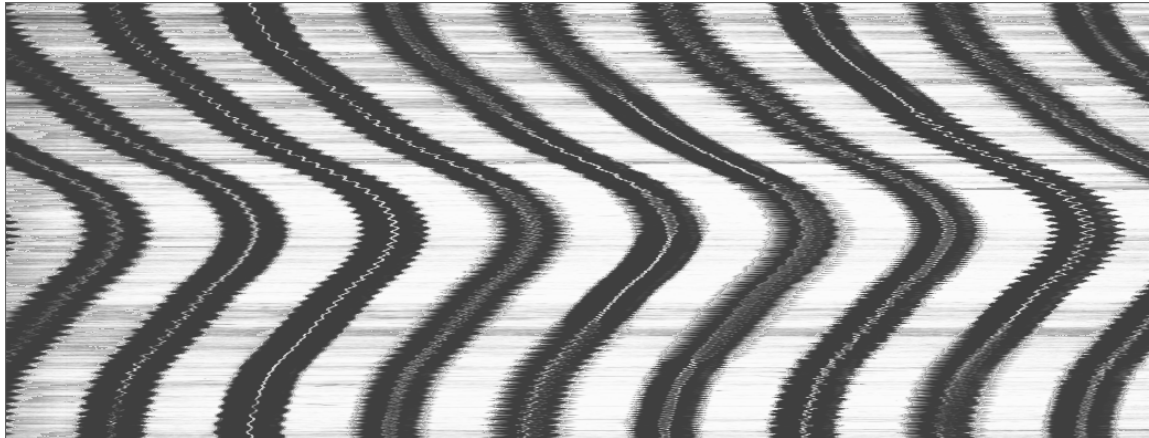


Figure 36: Vertically compressed 80'000 by 4096 image. A global '>' shape can be seen when following the grooves, provoked by a slightly off-center hole.

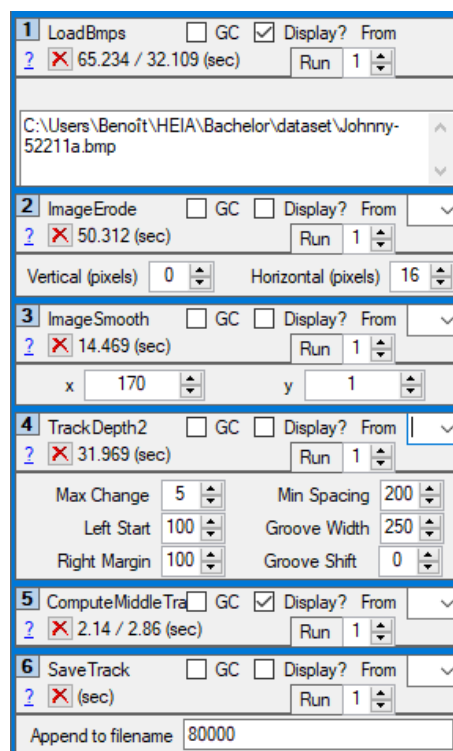


Figure 37: Chain of Weaver plugins and their parameters used for edge detection on disc images.

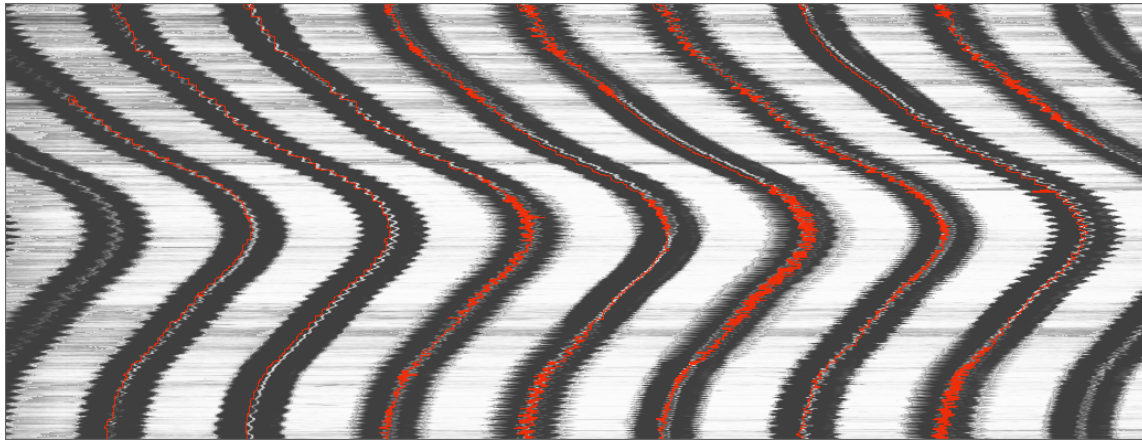


Figure 38: In red, track representing the middle of the grooves, found by Weaver.

8.2 Direct prediction

First, let's see what happens when we try to predict the audio from the image using a model trained on simple noisy grooves.

The real disc groove is actually wider than the ones used until this point. The real grooves are around 240 pixels wide, and the previous models used grooves of 160 pixels. It means a new model has to be trained with the correct dimensions. More noise is also added on each image, to make the model more robust to noise. The complete list of Weaver generated grooves can be found in the appendices.

Following the same idea as prototype 2, the images used for training are noised on runtime and the ones used for testing are noised and stored before, so that tests can be compared (see figure 39 for an example). The properties of each test image, the parameters of the noising function, and the seed used for randomization are given in the appendices.

The model hyper-parameters are the one that obtained the best results in prototype 2, except for the sliding window width. It should cover all four edges of the groove. To compensate off-centered tracking points, it needs to be larger than the groove width : 320 pixels seems good. After training for a few epochs, results are compared (figure 40) to select the best version.

Epoch 4 gives the best result over all test images, never seen during training. A prediction is made with this model on one of the test image. Figure 41 shows the predicted audio wave.

It looks like a good prediction.

Only one 80'000 by 4096 image will be used, as it represents around 5 seconds of

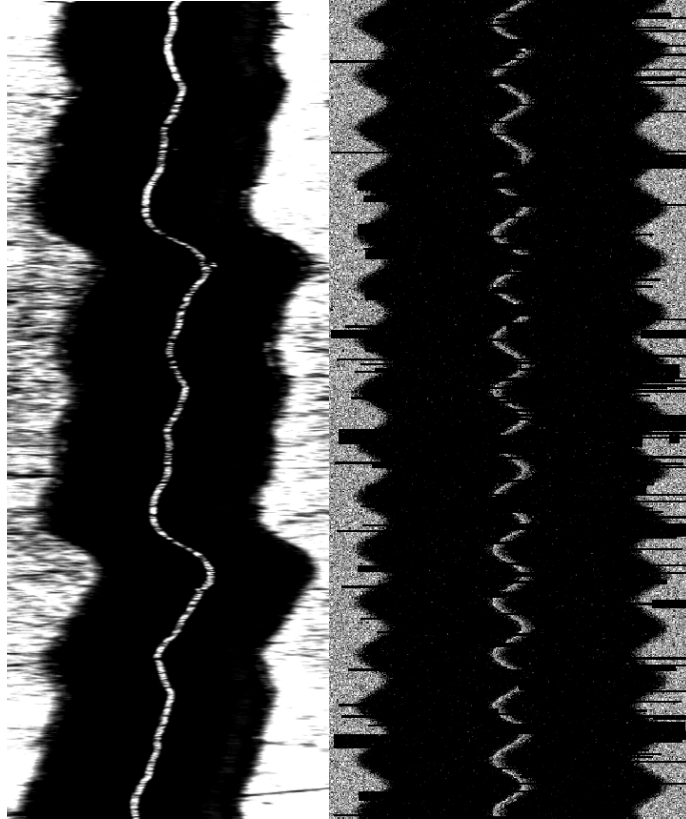


Figure 39: Right : Example of a noisy image used for testing. Left : Actual disc groove

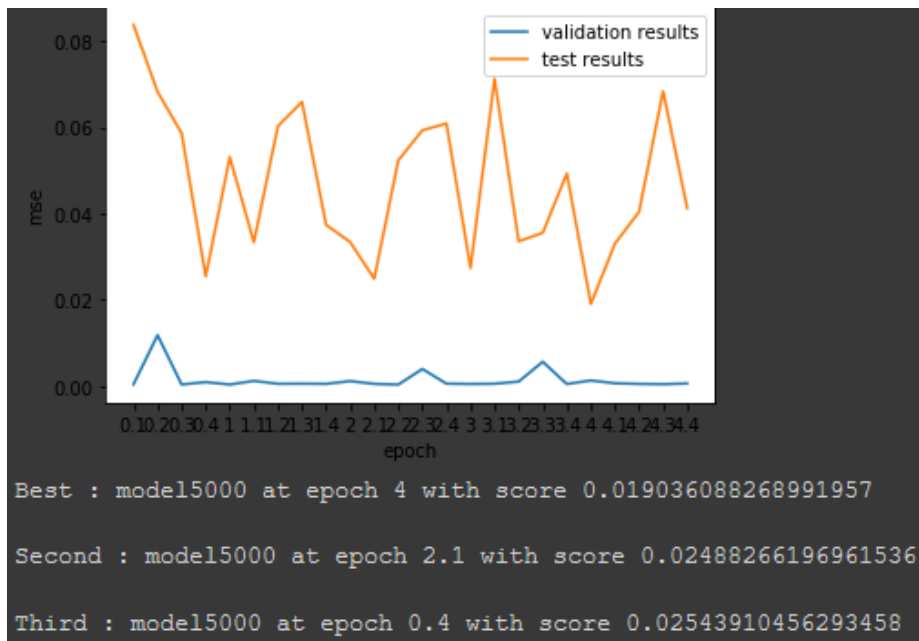


Figure 40: Graph of the validation and test results obtained

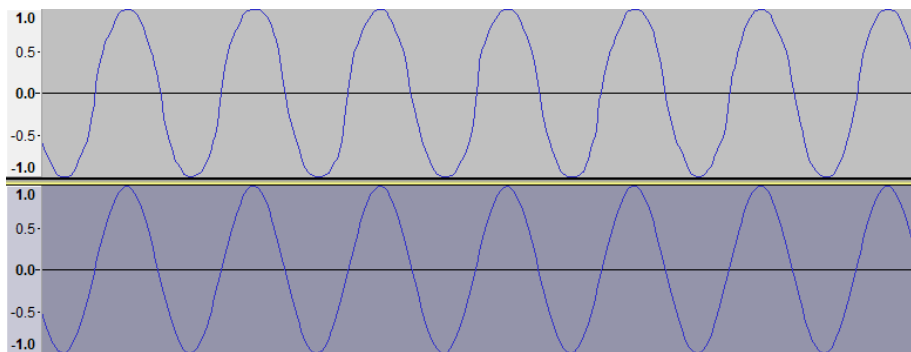


Figure 41: Top : predicted audio wave of a noisy 1850 Hz groove. Bottom : perfect audio signal of 1850 Hz.

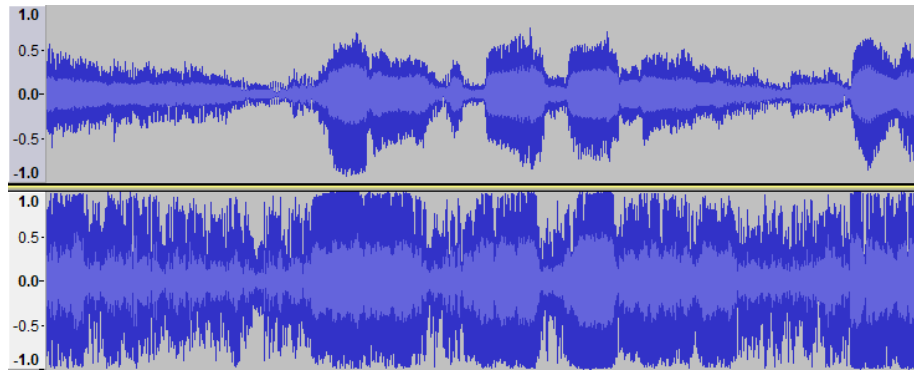


Figure 42: Top : Audio from tape. Bottom : Predicted audio using disc image

audio data. It is consequent enough for what we are trying to obtain.

The chosen image is called Johnny-52211 in the LBNL database.

Using the disc groove track generated with Weaver, sliding windows are given to the model for prediction. For this image, there are more than 550'000 windows to predict from. Figure 42 shows the audio wave predicted for the image compared to the actual audio from the tape.

We can already see that there is a lot of noise. The spectrums in figures 43 and 44 show that the noise is spread on all frequencies.

Trying to directly make predictions in a model that has never seen disc images doesn't give good results. The next step is to train the model further using disc images.

8.3 Transfer learning

Transfer learning is a method used in machine learning for problems similar to other already known problems. For example if you need to classify animals into horse or lama, and you have a access to a model that was trained for cats and dogs classification, transfer learning can probably be used.

The main idea is to use the already trained weights as initial values for your model. If the input or output shape are different, the trained model can be wrapped by new layers to accept the new shapes.

In our case, the first model was made so that it would already take the right shape of input. The model can simply be loaded in the software and further trained with disc images.

The main problem comes from giving the goal to the model. The clean tape version of the sound wave is available, but there is no way to know exactly which

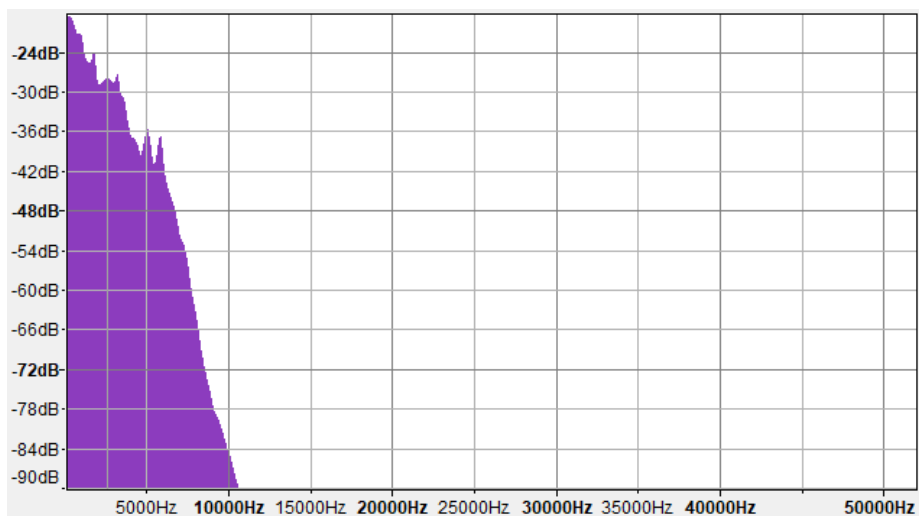


Figure 43: Spectrum of the tape audio corresponding to the Johnny-52211 image

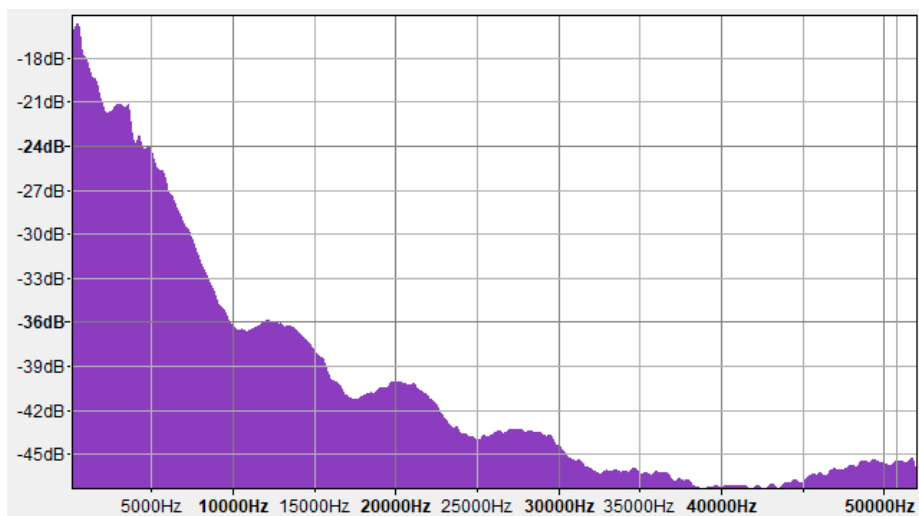


Figure 44: Spectrum of the predicted audio wave on image Johnny-52211

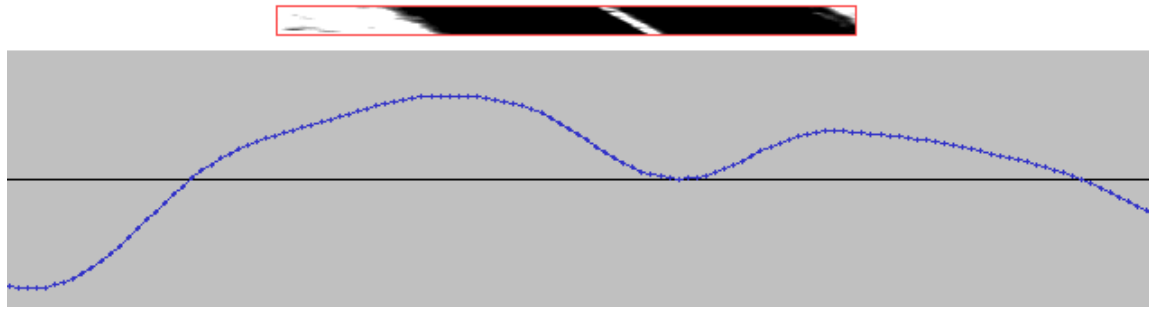


Figure 45: Inside the red rectangle : one window of pixels given to the model. Under : the audio wave from the tape. Only 15 of the audio data points match the window : which ones ?

point of the groove corresponds to which audio wave sample. As explained in the 8.1 Dataset section, one way to do it is to manually align the clean sound with a Weaver generated one.

However, it is far from easy. The Weaver generated audio wave is imperfect : it has a bigger amplitude range, high frequency waves, and a different shape on most sections, making impossible to perfectly align. Our model takes a window of pixels as input, and one amplitude as goal : we need the amplitude to corresponds to one of the line in the window. Figure 45 shows what the data looks like.

After manually aligning the Weaver generated audio to the tape audio as best as possible, the tape audio is cut to have the same amount of data points as the track of the groove. Since it is not sure that everything is matching correctly, only one image is used to train the model. We at least want good results on validation data (15% of the windows are not given to the model for training and are used after to evaluate it). Since there is a lot of windows (more than 550'000), the model is trained little by little to avoid running out of RAM. Only 50'000 windows are loaded at a time.

After training on the entire image 10 times, a sound prediction of the same image is made. Audio wave can be seen in figure 46, and its spectrum in figure 47.

Again, the results are not very good. When listening to the predicted sound, the singer can be heard, but there is a lot of noise on top of it.

It may be because of the misalignment of the goal audio wave regarding the groove track, or bad model hyper-parameters since the data is slightly different. It may also be that transfer learning is a bad idea for this problem, and the data are more different than it seems.

Our last hope for now is to train a new model only on disc images.

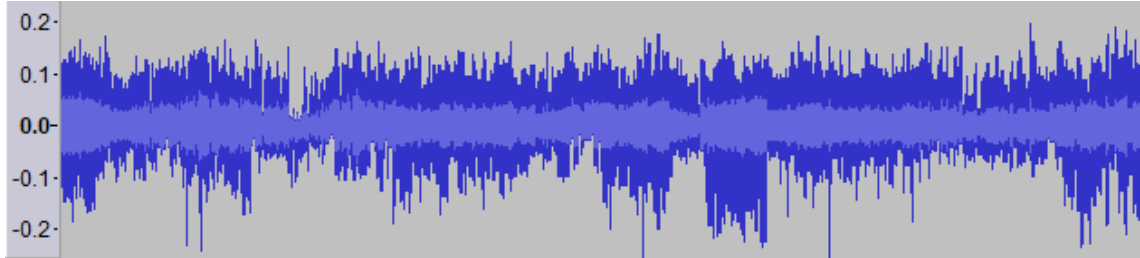


Figure 46: Audio wave predicted by the model trained first on simple noised grooves, then on disc image.

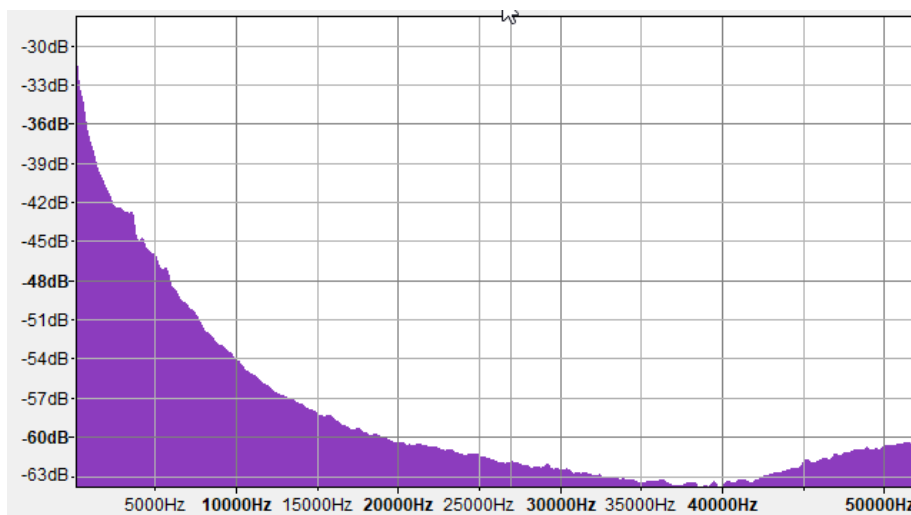


Figure 47: Spectrum of the sound wave predicted by the model trained first on simple noised grooves, then on disc image.

8.4 Training new model

The dataset preparation is exactly the same as for transfer learning. Creating a new model and training it can be tried immediately. Again, since the perfect alignment of goal audio and groove track is tricky, only one image is used at first to validate the models.

In order to account for the possible misalignment of the audio wave, the sliding window uses more lines of pixels : 21 now, against 15 before. Making the window even bigger would allow for more errors during alignment, but the model would need to learn to ignore a lot of lines that are far from the interesting ones. Furthermore, the window width is also increased from 320 to 360 pixels, as the track is sometimes quite off-center, and one of the edges is partially or entirely outside of the window.

Same as before, 15% of the windows are ignored during training, and used for validation scoring.

As first fine-tuning, different learning rates are tried , from $1e-8$ to 10 exponentially. All other parameters are kept the same as the best model from prototype 2.

After training for 3 times on the image, some values can already be removed for the possibilities : learning rates from $1e-2$ to 10 prevents the model from learning anything. The MSE is stuck around 1.0 at all times for those learning rates, which is really bad considering that all our values are normalized.

The other versions are trained until epoch 6. However, none of them can produce a nice sound wave. They all predict a line close to 0 amplitude for some reason.

Also, it has come to attention that the distortion of the groove image due to the off-center hole causes the Weaver generated audio to be faster at certain places and slower at others when comparing with the tape version. A possible workaround would be to align the audio in smaller parts, but it would take a lot of time.

9 Conclusion

The ML4NR project was not able to improve the current disc image process for sound wave generation. However, it obtained promising results when dealing with a simplified version of the problem. It paves the road for future projects going in the same direction. Prototypes 1 and 2 showed that a CNN can quickly learn to find lateral movement in a groove image, even when it is noised. There is no reason this ability can't be used for actual disc images.

Better results can certainly be obtained with further fine-tuning. All hyperparameters affect the others in some way, and finding the right combination deserves

a project in itself.

9.1 Personal conclusion

I really liked working on the ML4NR project. I learned a lot regarding machine learning, and everything was interesting.

The working environment was good, as all researchers at the LBNL were very nice.

I sometimes felt overwhelmed by the amount of possibilities offered by machine learning techniques. Everything seemed doable, and I was lost at what to try. Fortunately, my supervisors and consultants guided me in the right direction every time.

A citation I found during my machine learning researches stuck with me : "Determining the topology/ flavor/ training method/ hyper parameters for deep learning is a black art with not much theory to guide you." [7]

9.2 Future work

Multiple leads on how to possibly obtain better results emerged during the project :

- Generate a "better" artificial noise for simple grooves. Do a detailed analysis on the actual noise found on disc images, and replicate it on clean artificial groove images. For example, the Gaussian noise is mainly found on the edges and not on the entire image.
- Test other optimizers and their properties. Finding the right optimizer and its correct parameters depends on the other hyper-parameters, so it should be done last.
- Prepare a bigger real disc images dataset, with correctly aligned audio goal. This could be a project on its own, because a lot of manipulations need to be done case by case.
- Going into a completely different direction, the audio generation could be improved by removing the noise from disc images. Auto-encoders seem to have good results for this type of problem. It would also need a detailed analysis of the actual noise.
- Other neural networks can be tried on this problem, for example LSTM.

10 Declaration of honor

I, the undersigned Benoit Ruffray, hereby declare on my honor that this document is the result of my own work. I certify that I have not cheated by means of plagiarism or any other forms of fraud. All my information sources and all author citations have been clearly mentioned.

11 References

- [1] Wikipedia. Sound recording and reproduction : Phonograph. https://en.wikipedia.org/wiki/Sound_recording_and_reproduction#Phonograph, jun 2019.
- [2] Humanity. Tensorflow. <https://github.com/tensorflow/tensorflow>, jun 2019.
- [3] Keras team. Keras. <https://keras.io/>, jun 2019.
- [4] Alex Krizhevsky Vinod Nair Geoffrey Hinton. Cifar-10 homepage. <https://www.cs.toronto.edu/~kriz/cifar.html>, jun 2019.
- [5] Leonardo Araujo dos Santos. Artificial intelligence - machine learning. https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/machine_learning.html, aug 2019.
- [6] Jason Brownlee. Multivariate time series forecasting with lstms in keras. <https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/>, aug 2019.
- [7] Prakash Jay. Transfer learning using keras. <https://medium.com/@14prakash/transfer-learning-using-keras-d804b2e04ef8>, aug 2019.

Appendices

A Prototype 1

A.1 Models and results

All models, results and parameters can be found on the Google Drive folder.

```

model = Sequential()
model.add(Conv2D(32, filter_shapes[0],
                 input_shape=(time_steps, img_width, 1), padding=padding))
model.add(Activation(activation_function))
model.add(Conv2D(32, filter_shapes[1], padding=padding))
model.add(Activation(activation_function))
model.add(MaxPooling2D(pool_size=max_pool_shape))
model.add(Dropout(dropout_rate))

model.add(Conv2D(64, filter_shapes[2], padding=padding))
model.add(Activation(activation_function))
model.add(Conv2D(64, filter_shapes[3], padding=padding))
model.add(Activation(activation_function))
model.add(MaxPooling2D(pool_size=max_pool_shape))
model.add(Dropout(dropout_rate))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation(activation_function))
model.add(Dropout(dropout_rate*2))
model.add(Dense(predictions))
model.add(Activation('tanh'))

opt = keras.optimizers.rmsprop(lr=learning_rate, decay=decay)

model.compile(loss='mean_squared_error',
              optimizer=opt,
              metrics=['mean_absolute_error', 'mean_squared_error'])

```

Figure 48: Code used to create models for prototype 1. All models follow this architecture.

Model name	time steps	pred	conv height	conv width	pool height	pool width	drop rate	learn rate	decay rate	best MSE	epoch
model1	5	1	3	3	2	2	0.25	0.0001	1e-06	0.0069	1
model2	5	1	3	5	2	2	0.25	0.0001	1e-06	0.0095	1
model3	5	1	3	7	2	2	0.25	0.0001	1e-06	0.0088	1
model4	5	1	3	9	2	2	0.25	0.0001	1e-06	0.0082	1
model5	5	3	3	3	2	2	0.25	0.0001	1e-06	0.0196	1
model6	5	3	3	5	2	2	0.25	0.0001	1e-06	0.0137	1
model7	5	3	3	7	2	2	0.25	0.0001	1e-06	0.0147	1
model8	5	3	3	9	2	2	0.25	0.0001	1e-06	0.0144	1
model9	5	5	3	3	2	2	0.25	0.0001	1e-06	0.0298	1
model10	5	5	3	5	2	2	0.25	0.0001	1e-06	0.0214	1
model11	5	5	3	7	2	2	0.25	0.0001	1e-06	0.0242	1
model12	5	5	3	9	2	2	0.25	0.0001	1e-06	0.0208	1
model13	7	1	3	3	2	2	0.25	0.0001	1e-06	0.0244	1
model14	7	1	3	5	2	2	0.25	0.0001	1e-06	0.0227	1
model15	7	1	3	7	2	2	0.25	0.0001	1e-06	0.0343	1
model16	7	1	3	9	2	2	0.25	0.0001	1e-06	0.0344	1

Model name	time steps	pred	conv height	conv width	pool height	pool width	drop rate	learn rate	decay rate	best MSE	epoch
model17	7	1	4	3	2	2	0.25	0.0001	1e-06	0.0215	1
model18	7	1	4	5	2	2	0.25	0.0001	1e-06	0.0392	1
model19	7	1	4	7	2	2	0.25	0.0001	1e-06	0.0356	1
model20	7	1	4	9	2	2	0.25	0.0001	1e-06	0.0353	1
model21	7	3	3	3	2	2	0.25	0.0001	1e-06	0.0383	1
model22	7	3	3	5	2	2	0.25	0.0001	1e-06	0.0340	1
model23	7	3	3	7	2	2	0.25	0.0001	1e-06	0.0385	1
model24	7	3	3	9	2	2	0.25	0.0001	1e-06	0.0372	1
model25	7	3	4	3	2	2	0.25	0.0001	1e-06	0.0349	1
model26	7	3	4	5	2	2	0.25	0.0001	1e-06	0.0268	1
model27	7	3	4	7	2	2	0.25	0.0001	1e-06	0.0350	1
model28	7	3	4	9	2	2	0.25	0.0001	1e-06	0.0370	1
model29	7	5	3	3	2	2	0.25	0.0001	1e-06	0.0424	1
model30	7	5	3	5	2	2	0.25	0.0001	1e-06	0.0468	1
model31	7	5	3	7	2	2	0.25	0.0001	1e-06	0.0415	1
model32	7	5	3	9	2	2	0.25	0.0001	1e-06	0.0444	1
model33	7	5	4	3	2	2	0.25	0.0001	1e-06	0.0412	1
model34	7	5	4	5	2	2	0.25	0.0001	1e-06	0.0465	1
model35	7	5	4	7	2	2	0.25	0.0001	1e-06	0.0437	1
model36	7	5	4	9	2	2	0.25	0.0001	1e-06	0.0395	1
model37	7	7	3	3	2	2	0.25	0.0001	1e-06	0.0577	1
model38	7	7	3	5	2	2	0.25	0.0001	1e-06	0.0536	1
model39	7	7	3	7	2	2	0.25	0.0001	1e-06	0.0561	1
model40	7	7	3	9	2	2	0.25	0.0001	1e-06	0.0530	1
model41	7	7	4	3	2	2	0.25	0.0001	1e-06	0.0510	1
model42	7	7	4	5	2	2	0.25	0.0001	1e-06	0.0477	1
model43	7	7	4	7	2	2	0.25	0.0001	1e-06	0.0571	1
model44	7	7	4	9	2	2	0.25	0.0001	1e-06	0.0568	1
model45	9	1	3	3	2	2	0.25	0.0001	1e-06	0.0288	1
model46	9	1	3	5	2	2	0.25	0.0001	1e-06	0.0303	1
model47	9	1	3	7	2	2	0.25	0.0001	1e-06	0.0344	1
model48	9	1	3	9	2	2	0.25	0.0001	1e-06	0.0282	1
model49	9	1	4	3	2	2	0.25	0.0001	1e-06	0.0270	1
model50	9	1	4	5	2	2	0.25	0.0001	1e-06	0.0315	1
model51	9	1	4	7	2	2	0.25	0.0001	1e-06	0.0285	1

Model name	time steps	pred	conv height	conv width	pool height	pool width	drop rate	learn rate	decay rate	best MSE	epoch
model52	9	1	4	9	2	2	0.25	0.0001	1e-06	0.0248	1
model53	9	1	5	3	2	2	0.25	0.0001	1e-06	0.0242	1

Model name	time steps	pred	conv height	conv width	pool height	pool width	drop rate	learn rate	decay rate	best MSE	epoch
model1000	5	1	3	3	1	1	0.0	0.0001	1e-06	0.0554	1

A.2 Image dataset

Training dataset
f110_a10_e3_b12_g160_h80000.tif
f1150_a10_e3_b12_g160_h80000.tif
f1155_a10_e3_b12_g160_h80000.tif
f1305_a10_e3_b12_g160_h80000.tif
f1355_a10_e3_b12_g160_h80000.tif
f1400_a10_e3_b12_g160_h80000.tif
f1480_a10_e3_b12_g160_h80000.tif
f1510_a10_e3_b12_g160_h80000.tif
f1765_a10_e3_b12_g160_h80000.tif
f1815_a10_e3_b12_g160_h80000.tif
f1875_a10_e3_b12_g160_h80000.tif
f1935_a10_e3_b12_g160_h80000.tif
f1955_a10_e3_b12_g160_h80000.tif
f2035_a10_e3_b12_g160_h80000.tif
f2075_a10_e3_b12_g160_h80000.tif
f210_a10_e3_b12_g160_h80000.tif
f2110_a10_e3_b12_g160_h80000.tif
f2150_a10_e3_b12_g160_h80000.tif
f2190_a10_e3_b12_g160_h80000.tif
f2210_a10_e3_b12_g160_h80000.tif
f2260_a10_e3_b12_g160_h80000.tif
f2330_a10_e3_b12_g160_h80000.tif
f235_a10_e3_b12_g160_h80000.tif
f2390_a10_e3_b12_g160_h80000.tif

f2420_a10_e3_b12_g160_h80000.tif
f2470_a10_e3_b12_g160_h80000.tif
f2490_a10_e3_b12_g160_h80000.tif
f2580_a10_e3_b12_g160_h80000.tif
f25_a10_e3_b12_g160_h80000.tif
f2740_a10_e3_b12_g160_h80000.tif
f2760_a10_e3_b12_g160_h80000.tif
f2775_a10_e3_b12_g160_h80000.tif
f3045_a10_e3_b12_g160_h80000.tif
f3050_a10_e3_b12_g160_h80000.tif
f3125_a10_e3_b12_g160_h80000.tif
f3140_a10_e3_b12_g160_h80000.tif
f3200_a10_e3_b12_g160_h80000.tif
f3205_a10_e3_b12_g160_h80000.tif
f3220_a10_e3_b12_g160_h80000.tif
f3365_a10_e3_b12_g160_h80000.tif
f3405_a10_e3_b12_g160_h80000.tif
f350_a10_e3_b12_g160_h80000.tif
f3530_a10_e3_b12_g160_h80000.tif
f3790_a10_e3_b12_g160_h80000.tif
f3795_a10_e3_b12_g160_h80000.tif
f3810_a10_e3_b12_g160_h80000.tif
f3820_a10_e3_b12_g160_h80000.tif
f3825_a10_e3_b12_g160_h80000.tif
f3900_a10_e3_b12_g160_h80000.tif
f3935_a10_e3_b12_g160_h80000.tif
f3940_a10_e3_b12_g160_h80000.tif
f4055_a10_e3_b12_g160_h80000.tif
f4060_a10_e3_b12_g160_h80000.tif
f4215_a10_e3_b12_g160_h80000.tif
f435_a10_e3_b12_g160_h80000.tif
f4390_a10_e3_b12_g160_h80000.tif
f4505_a10_e3_b12_g160_h80000.tif
f4670_a10_e3_b12_g160_h80000.tif
f4780_a10_e3_b12_g160_h80000.tif
f4785_a10_e3_b12_g160_h80000.tif

f4875_a10_e3_b12_g160_h80000.tif
f4885_a10_e3_b12_g160_h80000.tif
f4950_a10_e3_b12_g160_h80000.tif
f4955_a10_e3_b12_g160_h80000.tif
f4990_a10_e3_b12_g160_h80000.tif
f50_a10_e3_b12_g160_h80000.tif
f510_a10_e3_b12_g160_h80000.tif
f545_a10_e3_b12_g160_h80000.tif
f555_a10_e3_b12_g160_h80000.tif
f595_a10_e3_b12_g160_h80000.tif
f600_a10_e3_b12_g160_h80000.tif
f650_a10_e3_b12_g160_h80000.tif
f710_a10_e3_b12_g160_h80000.tif
f75_a10_e3_b12_g160_h80000.tif
f765_a10_e3_b12_g160_h80000.tif
f905_a10_e3_b12_g160_h80000.tif
f925_a10_e3_b12_g160_h80000.tif
f975_a10_e3_b12_g160_h80000.tif

Table 5: Training dataset of prototype 1

Each file name contains the parameters used to generate it in the Weaver plugin. For example, f1875_a10_e3_b12_g160_h80000.tif is a groove of frequency 1875 Hz, amplitude 10, edge width 3, bottom width 12, groove width 160, and 80'000 pixels of height.

Testing dataset
f1550_a10_e3_b12_g160_h80000.tif
f2100_a10_e3_b12_g160_h80000.tif
f2175_a10_e3_b12_g160_h80000.tif
f2375_a10_e3_b12_g160_h80000.tif
f2695_a10_e3_b12_g160_h80000.tif
f2865_a10_e3_b12_g160_h80000.tif
f3170_a10_e3_b12_g160_h80000.tif
f3255_a10_e3_b12_g160_h80000.tif
f3540_a10_e3_b12_g160_h80000.tif
f3875_a10_e3_b12_g160_h80000.tif

f405_a10_e3_b12_g160_h80000.tif
f4200_a10_e3_b12_g160_h80000.tif
f4580_a10_e3_b12_g160_h80000.tif
f4845_a10_e3_b12_g160_h80000.tif
f4960_a10_e3_b12_g160_h80000.tif
f60_a10_e3_b12_g160_h80000.tif
f620_a10_e3_b12_g160_h80000.tif
f965_a10_e3_b12_g160_h80000.tif

Table 6: Testing dataset of prototype 1

B Prototype 2

B.1 Models and results

All models, results and parameters can be found on the Google Drive folder.

Model name	time steps	pred	conv height	conv width	pool height	pool width	drop rate	learn rate	decay rate	best MSE	epoch
model3000	5	1	3	3	2	2	0.25	0.0001	1e-06	0.0311	5
model3001	5	1	3	3	2	2	0.25	0.001	1e-05	0.0321	1
model3002	5	1	3	3	2	2	0.25	0.001	0.0	0.0387	5
model3003	15	1	3	3	2	2	0.25	0.001	0.0	0.0254	4
model3004	9	1	3	3	2	2	0.25	0.001	0.0	0.0323	1

Model name	time steps	pred	conv height	conv width	pool height	pool width	drop rate	learn rate	decay rate	best MSE	epoch
model4000	15	1	3	3	2	2	0.25	0.001	0.0	0.0287	5
model4001	5	1	3	3	2	2	0.25	0.001	0.0	0.0405	0.8
model4002	15	1	3	3	2	2	0.25	0.001	0.0	0.0268	1
model4003	15	1	3	3	2	2	0.25	0.01	0.0	1.5004	0.1
model4004	15	1	3	3	2	2	0.25	0.001	0.0	0.0172	2.6
model4005	5	1	3	3	2	2	0.25	0.001	0.0	0.0346	1.6
model4006	25	1	3	3	2	2	0.25	0.001	0.0	0.0294	0.1

B.2 Image dataset

Testing dataset
f1140_a10_e4_b10_g160_h80000.tif
f1294_a10_e2_b10_g150_h80000.tif
f1530_a10_e2_b10_g155_h80000.tif
f1681_a10_e3_b13_g150_h80000.tif
f173_a10_e3_b13_g165_h80000.tif
f1862_a10_e3_b13_g165_h80000.tif
f1976_a10_e2_b12_g165_h80000.tif
f2042_a10_e2_b12_g160_h80000.tif
f2161_a10_e3_b10_g160_h80000.tif
f2426_a10_e2_b10_g150_h80000.tif
f2626_a10_e4_b13_g150_h80000.tif
f2709_a10_e3_b11_g165_h80000.tif
f2905_a10_e3_b11_g165_h80000.tif
f294_a10_e3_b10_g165_h80000.tif
f31_a10_e4_b12_g150_h80000.tif
f3202_a10_e3_b11_g160_h80000.tif
f3411_a10_e3_b10_g160_h80000.tif
f3577_a10_e4_b12_g155_h80000.tif
f3817_a10_e4_b12_g155_h80000.tif
f3968_a10_e4_b12_g165_h80000.tif
f4070_a10_e4_b12_g155_h80000.tif
f426_a10_e4_b11_g155_h80000.tif
f4299_a10_e2_b11_g160_h80000.tif
f4452_a10_e4_b13_g150_h80000.tif
f4560_a10_e4_b13_g160_h80000.tif
f4742_a10_e2_b10_g160_h80000.tif
f4880_a10_e4_b11_g150_h80000.tif
f610_a10_e4_b13_g165_h80000.tif
f673_a10_e2_b13_g155_h80000.tif
f862_a10_e2_b13_g160_h80000.tif

Training dataset
f1034_a10_e2_b13_g150_h80000.tif

f1036_a10_e2_b12_g150_h80000.tif
f1043_a10_e2_b13_g165_h80000.tif
f1159_a10_e2_b10_g165_h80000.tif
f1227_a10_e2_b10_g155_h80000.tif
f1231_a10_e2_b10_g155_h80000.tif
f1236_a10_e4_b13_g155_h80000.tif
f1245_a10_e4_b13_g150_h80000.tif
f1286_a10_e4_b13_g155_h80000.tif
f1327_a10_e2_b13_g155_h80000.tif
f1356_a10_e2_b11_g165_h80000.tif
f1385_a10_e3_b11_g165_h80000.tif
f144_a10_e2_b10_g150_h80000.tif
f1492_a10_e2_b12_g155_h80000.tif
f150_a10_e3_b12_g160_h80000.tif
f1519_a10_e4_b12_g150_h80000.tif
f1522_a10_e2_b13_g165_h80000.tif
f158_a10_e4_b11_g165_h80000.tif
f1612_a10_e2_b10_g155_h80000.tif
f1643_a10_e2_b13_g165_h80000.tif
f1645_a10_e2_b11_g165_h80000.tif
f1660_a10_e2_b11_g155_h80000.tif
f1667_a10_e3_b11_g155_h80000.tif
f1675_a10_e3_b13_g155_h80000.tif
f1742_a10_e4_b11_g150_h80000.tif
f177_a10_e4_b12_g160_h80000.tif
f1784_a10_e3_b12_g165_h80000.tif
f1802_a10_e2_b13_g150_h80000.tif
f1802_a10_e4_b13_g160_h80000.tif
f1818_a10_e3_b12_g150_h80000.tif
f1842_a10_e2_b11_g165_h80000.tif
f1927_a10_e2_b12_g160_h80000.tif
f1934_a10_e3_b11_g165_h80000.tif
f1975_a10_e4_b10_g165_h80000.tif
f1977_a10_e4_b13_g165_h80000.tif
f1996_a10_e4_b13_g150_h80000.tif
f2081_a10_e4_b11_g150_h80000.tif

f2084_a10_e3_b10_g160_h80000.tif
f2085_a10_e2_b11_g165_h80000.tif
f2096_a10_e3_b13_g150_h80000.tif
f2148_a10_e2_b13_g150_h80000.tif
f2156_a10_e4_b10_g150_h80000.tif
f2204_a10_e2_b10_g150_h80000.tif
f220_a10_e3_b12_g155_h80000.tif
f2220_a10_e2_b12_g155_h80000.tif
f2236_a10_e3_b10_g155_h80000.tif
f2239_a10_e4_b13_g165_h80000.tif
f2319_a10_e4_b12_g150_h80000.tif
f2384_a10_e3_b10_g150_h80000.tif
f2426_a10_e4_b11_g160_h80000.tif
f2438_a10_e3_b12_g160_h80000.tif
f2461_a10_e2_b11_g155_h80000.tif
f2475_a10_e3_b12_g160_h80000.tif
f2479_a10_e4_b11_g165_h80000.tif
f247_a10_e2_b12_g165_h80000.tif
f2530_a10_e3_b10_g155_h80000.tif
f2630_a10_e3_b12_g160_h80000.tif
f2642_a10_e3_b13_g150_h80000.tif
f2672_a10_e4_b11_g155_h80000.tif
f2673_a10_e4_b12_g150_h80000.tif
f2695_a10_e2_b10_g155_h80000.tif
f2708_a10_e2_b13_g160_h80000.tif
f272_a10_e2_b10_g150_h80000.tif
f273_a10_e3_b13_g160_h80000.tif
f2811_a10_e4_b12_g165_h80000.tif
f2816_a10_e3_b11_g160_h80000.tif
f2827_a10_e4_b10_g160_h80000.tif
f2829_a10_e2_b10_g165_h80000.tif
f2878_a10_e4_b13_g150_h80000.tif
f2889_a10_e2_b11_g155_h80000.tif
f291_a10_e4_b11_g165_h80000.tif
f2971_a10_e2_b13_g150_h80000.tif
f2977_a10_e2_b13_g165_h80000.tif

f3004_a10_e2_b10_g160_h80000.tif
f3029_a10_e3_b10_g165_h80000.tif
f3055_a10_e2_b11_g160_h80000.tif
f3129_a10_e2_b12_g165_h80000.tif
f3280_a10_e2_b13_g165_h80000.tif
f3292_a10_e4_b11_g165_h80000.tif
f3307_a10_e3_b11_g150_h80000.tif
f3318_a10_e2_b11_g155_h80000.tif
f3348_a10_e3_b11_g155_h80000.tif
f3383_a10_e4_b11_g155_h80000.tif
f3415_a10_e2_b13_g155_h80000.tif
f3426_a10_e3_b12_g165_h80000.tif
f3428_a10_e4_b12_g150_h80000.tif
f3436_a10_e3_b10_g150_h80000.tif
f3473_a10_e3_b13_g160_h80000.tif
f3522_a10_e2_b13_g150_h80000.tif
f3605_a10_e4_b13_g160_h80000.tif
f3729_a10_e2_b12_g160_h80000.tif
f3734_a10_e3_b11_g165_h80000.tif
f3737_a10_e4_b13_g155_h80000.tif
f374_a10_e3_b12_g155_h80000.tif
f3754_a10_e4_b10_g160_h80000.tif
f3790_a10_e2_b10_g160_h80000.tif
f385_a10_e3_b11_g150_h80000.tif
f3903_a10_e4_b13_g155_h80000.tif
f3904_a10_e3_b13_g165_h80000.tif
f3905_a10_e3_b12_g155_h80000.tif
f3927_a10_e3_b11_g165_h80000.tif
f3930_a10_e2_b10_g165_h80000.tif
f3938_a10_e4_b10_g155_h80000.tif
f398_a10_e2_b11_g165_h80000.tif
f4017_a10_e2_b10_g165_h80000.tif
f4026_a10_e2_b11_g165_h80000.tif
f4040_a10_e3_b13_g160_h80000.tif
f4056_a10_e4_b10_g165_h80000.tif
f4063_a10_e2_b12_g155_h80000.tif

f4069_a10_e2_b12_g150_h80000.tif
f4072_a10_e3_b12_g155_h80000.tif
f4094_a10_e2_b13_g150_h80000.tif
f409_a10_e2_b12_g160_h80000.tif
f415_a10_e2_b10_g160_h80000.tif
f4179_a10_e2_b12_g150_h80000.tif
f4184_a10_e2_b13_g165_h80000.tif
f420_a10_e2_b10_g150_h80000.tif
f4242_a10_e4_b13_g150_h80000.tif
f4243_a10_e4_b11_g150_h80000.tif
f4320_a10_e3_b13_g160_h80000.tif
f4328_a10_e3_b13_g165_h80000.tif
f4345_a10_e3_b13_g155_h80000.tif
f4366_a10_e4_b13_g155_h80000.tif
f4388_a10_e2_b11_g150_h80000.tif
f4410_a10_e4_b12_g160_h80000.tif
f4459_a10_e4_b13_g165_h80000.tif
f4476_a10_e2_b10_g165_h80000.tif
f4499_a10_e2_b11_g165_h80000.tif
f4512_a10_e3_b10_g155_h80000.tif
f4534_a10_e4_b11_g155_h80000.tif
f4551_a10_e4_b10_g160_h80000.tif
f4611_a10_e2_b11_g160_h80000.tif
f4650_a10_e2_b11_g150_h80000.tif
f4700_a10_e2_b13_g160_h80000.tif
f4711_a10_e2_b10_g165_h80000.tif
f4740_a10_e2_b12_g160_h80000.tif
f4740_a10_e2_b12_g165_h80000.tif
f4745_a10_e2_b13_g165_h80000.tif
f4750_a10_e4_b10_g165_h80000.tif
f4771_a10_e3_b10_g165_h80000.tif
f4771_a10_e4_b12_g155_h80000.tif
f4772_a10_e4_b11_g165_h80000.tif
f4779_a10_e4_b10_g160_h80000.tif
f47_a10_e3_b10_g150_h80000.tif
f485_a10_e2_b13_g160_h80000.tif

f4969_a10_e3_b13_g165_h80000.tif
f4977_a10_e2_b10_g155_h80000.tif
f4978_a10_e4_b12_g150_h80000.tif
f497_a10_e4_b12_g155_h80000.tif
f528_a10_e2_b12_g155_h80000.tif
f533_a10_e3_b13_g155_h80000.tif
f558_a10_e3_b13_g150_h80000.tif
f604_a10_e3_b13_g155_h80000.tif
f616_a10_e4_b11_g150_h80000.tif
f617_a10_e3_b11_g160_h80000.tif
f645_a10_e3_b11_g155_h80000.tif
f666_a10_e4_b11_g160_h80000.tif
f666_a10_e4_b13_g155_h80000.tif
f671_a10_e2_b10_g155_h80000.tif
f678_a10_e3_b11_g150_h80000.tif
f705_a10_e2_b11_g160_h80000.tif
f728_a10_e2_b11_g160_h80000.tif
f754_a10_e4_b13_g160_h80000.tif
f77_a10_e3_b10_g160_h80000.tif
f827_a10_e4_b13_g160_h80000.tif
f84_a10_e2_b13_g165_h80000.tif
f853_a10_e3_b11_g165_h80000.tif
f870_a10_e2_b13_g150_h80000.tif
f874_a10_e2_b13_g160_h80000.tif
f942_a10_e3_b12_g165_h80000.tif

C Prototype 3 and 4

C.1 Models and results

All models, results and parameters can be found on the Google Drive folder.

Model 5000 : train only on artificial images

Model 5002+ : train only on disc, fin tune window height, dropout rate, learning rate.

Model 6001+ : fine tuning Adam optimizer

Model name	time steps	pred	conv height	conv width	pool height	pool width	drop rate	learn rate	decay rate	best MSE	
model5000	15	1	3	3	2	2	[0.25, 0.25, 0.5]	0.001	0.0	0.0000	
model5002	15	1	3	3	2	2	[0.25, 0.25, 0.5]	0.001	0.0	0.0000	
model5003	25	1	3	3	2	2	[0.25, 0.25, 0.5]	0.001	0.0	0.0000	
model5004	15	1	3	3	2	2	[0.5, 0.5, 0.7]	1	0.0	0.0000	
model5005	21	1	3	3	2	2	[0.5, 0.5, 0.7]	0.0005	1e-06	0.0000	

Model name	time steps	pred	conv height	conv width	pool height	pool width	drop rate	learn rate	decay rate	best MSE	ep
model6001	21	1	3	3	2	2	[0.25, 0.5, 0.5]	1e-08	0	0.0361	4
model6002	21	1	3	3	2	2	[0.25, 0.5, 0.5]	1e-07	0	0.0354	1
model6003	21	1	3	3	2	2	[0.25, 0.5, 0.5]	1e-06	0	0.0355	1
model6004	21	1	3	3	2	2	[0.25, 0.5, 0.5]	1e-05	0	0.0376	1
model6005	21	1	3	3	2	2	[0.25, 0.5, 0.5]	0.0001	0	0.0391	1
model6006	21	1	3	3	2	2	[0.25, 0.5, 0.5]	0.001	0	0.0353	3
model6007	21	1	3	3	2	2	[0.25, 0.5, 0.5]	0.01	0	1.0298	1
model6008	21	1	3	3	2	2	[0.25, 0.5, 0.5]	0.1	0	1.0298	1
model6009	21	1	3	3	2	2	[0.25, 0.5, 0.5]	1	0	1.0299	1
model6010	21	1	3	3	2	2	[0.25, 0.5, 0.5]	10	0	1.0298	1

C.2 Image dataset

Disc image : Johnny-52211.bmp

Testing dataset
f1080_a15_e13_b10_g237_h80000.tif
f1400_a14_e12_b10_g239_h80000.tif
f1660_a12_e14_b10_g241_h80000.tif
f1850_a14_e12_b11_g241_h80000.tif
f220_a14_e14_b10_g237_h80000.tif
f2250_a13_e14_b10_g243_h80000.tif
f2720_a17_e14_b10_g239_h80000.tif
f3600_a15_e12_b11_g235_h80000.tif
f3800_a13_e13_b10_g239_h80000.tif
f4020_a15_e14_b10_g235_h80000.tif
f4360_a15_e14_b10_g241_h80000.tif

f4650_a18_e12_b11_g239_h80000.tif
f4860_a15_e12_b10_g241_h80000.tif
f570_a17_e12_b11_g243_h80000.tif
f840_a16_e13_b11_g241_h80000.tif

Training dataset
f1050_a16_e14_b11_g235_h80000.tif
f1070_a17_e12_b10_g239_h80000.tif
f1100_a16_e12_b10_g239_h80000.tif
f1110_a18_e12_b11_g243_h80000.tif
f1210_a17_e13_b10_g235_h80000.tif
f1280_a11_e13_b10_g243_h80000.tif
f1420_a19_e13_b10_g243_h80000.tif
f1530_a16_e13_b11_g239_h80000.tif
f1590_a10_e13_b11_g239_h80000.tif
f1600_a16_e13_b11_g235_h80000.tif
f1690_a11_e14_b11_g237_h80000.tif
f1720_a11_e12_b11_g237_h80000.tif
f1720_a17_e13_b10_g241_h80000.tif
f1770_a12_e12_b10_g241_h80000.tif
f1810_a16_e14_b10_g237_h80000.tif
f1880_a19_e13_b10_g243_h80000.tif
f1950_a16_e12_b11_g239_h80000.tif
f1960_a14_e14_b11_g235_h80000.tif
f1980_a19_e14_b11_g237_h80000.tif
f2010_a11_e14_b11_g243_h80000.tif
f2230_a13_e14_b10_g241_h80000.tif
f2240_a15_e14_b10_g235_h80000.tif
f2320_a15_e12_b11_g243_h80000.tif
f2340_a16_e12_b11_g241_h80000.tif
f2340_a16_e14_b11_g239_h80000.tif
f2360_a13_e12_b11_g241_h80000.tif
f240_a16_e14_b11_g243_h80000.tif
f2420_a15_e12_b11_g239_h80000.tif
f2450_a19_e12_b10_g237_h80000.tif
f2510_a15_e13_b11_g235_h80000.tif

f2690_a11_e14_b10_g239_h80000.tif
f2690_a16_e13_b10_g239_h80000.tif
f270_a11_e13_b11_g235_h80000.tif
f280_a19_e13_b10_g237_h80000.tif
f3020_a13_e14_b10_g241_h80000.tif
f3110_a16_e14_b11_g237_h80000.tif
f3160_a17_e13_b11_g243_h80000.tif
f3220_a16_e13_b10_g235_h80000.tif
f3290_a15_e12_b10_g243_h80000.tif
f3450_a19_e13_b10_g243_h80000.tif
f350_a13_e14_b10_g241_h80000.tif
f3640_a10_e13_b11_g243_h80000.tif
f3640_a17_e14_b11_g237_h80000.tif
f3660_a19_e12_b11_g241_h80000.tif
f3690_a14_e13_b11_g239_h80000.tif
f370_a16_e14_b10_g235_h80000.tif
f3720_a19_e14_b11_g243_h80000.tif
f3790_a10_e13_b11_g237_h80000.tif
f3820_a13_e12_b10_g243_h80000.tif
f3840_a13_e13_b10_g241_h80000.tif
f3870_a17_e12_b10_g243_h80000.tif
f3930_a10_e13_b11_g239_h80000.tif
f4010_a14_e13_b10_g241_h80000.tif
f4030_a18_e14_b10_g235_h80000.tif
f4090_a10_e13_b11_g235_h80000.tif
f4170_a14_e12_b11_g237_h80000.tif
f4170_a17_e13_b11_g235_h80000.tif
f4340_a15_e14_b10_g237_h80000.tif
f440_a12_e14_b10_g235_h80000.tif
f4420_a14_e13_b10_g237_h80000.tif
f4440_a11_e12_b11_g237_h80000.tif
f4440_a13_e12_b10_g243_h80000.tif
f4470_a18_e12_b10_g243_h80000.tif
f4500_a15_e12_b11_g237_h80000.tif
f4560_a14_e13_b11_g241_h80000.tif
f4570_a18_e13_b11_g235_h80000.tif

f4590_a14_e12_b10_g237_h80000.tif
f4690_a15_e12_b10_g235_h80000.tif
f4690_a18_e13_b11_g243_h80000.tif
f4710_a11_e13_b10_g235_h80000.tif
f4710_a19_e14_b10_g237_h80000.tif
f4730_a17_e12_b11_g241_h80000.tif
f4750_a19_e13_b11_g243_h80000.tif
f4770_a16_e13_b10_g235_h80000.tif
f4770_a16_e13_b11_g241_h80000.tif
f4890_a15_e14_b10_g237_h80000.tif
f4910_a16_e13_b11_g235_h80000.tif
f610_a19_e13_b10_g239_h80000.tif
f660_a12_e14_b11_g239_h80000.tif
f720_a15_e13_b11_g239_h80000.tif
f790_a14_e12_b11_g241_h80000.tif
f810_a17_e13_b11_g237_h80000.tif
f850_a14_e14_b10_g243_h80000.tif
f890_a14_e12_b10_g241_h80000.tif
f950_a18_e12_b11_g243_h80000.tif

D Planning

Gantt Chart

