LAB University of Applied Sciences

Piano Note Recognition: Classification Aided by Convolutional Neural Networks

LAB University of Applied Sciences
Bachelor of Engineering, Information and Communication Technology
2021
Juan, Girbés Mínguez

Abstract

Author Girbés Mínguez, Juan	Publication type Thesis, UAS	Completion year 2021
	Number of pages 59	
Title of the thesis **Piano Note Recognition: Classification Aided by Convolutional Neural Network**		
Degree Bachelor of Engineering, Information and Communication Technology		
Name, title, and organisation of the thesis supervisor Ismo Jakonen, Senior Lecturer, Media Technology		

Abstract

Artificial Neural Networks has changed how we solve problems that seemed unsolvable. One of such problems is audio recognition. The aim of the thesis was to recognize piano musical notes using neural networks.

In this case, it was chosen the task of Note Identification, a task that most humans are not capable. It was tackled using a Convolutional Neural Network, as a problem of Supervised Learning. The Dataset was generated specially for this work, and in some cases, it was mixed with background rain noise. Even was expanded to the recognition of chords. Neural network efficiency was quantified as the recognition accuracy in unseen data.

A Convolutional Neural Network was created that recognized piano notes. Accuracy was 100% for one note, and 96.94% for 3-note chords with background noise.

In conclusion, Convolutional Neural Network allows the recognition of notes in environments with background noise. This approach could be used for more complex recognitions, including other instruments or sounds.

Keywords

Artificial Intelligence, Artificial Neural Network, Piano Notes, Note Recognition, Convolutional.

Contents

Acronyms

AI: Artificial Intelligence

ANN: Artificial Neural Network (ANNs in plural)

CNN: Convolutional Neural Network

ELU: Exponential Linear Unit Function

GAN: Generative Adversarial Network (GANs in plural)

LSTM: Long Short-Term Memory

NN: Neural Network (NNs in plural)

ReLU: Rectified Linear Unit Function

RReLU: Randomized Leaky Rectified Linear Unit

RNN: Recurrent Neural Networks (RNNs in plural)

SELU: Scale Exponential Linear Unit Function

SGD: Stochastic Gradient Descend

1 Introduction

1.1 Artificial Intelligence, Machine Learning, Deep Learning, and Neural Networks

Nowadays, the definitions of Artificial Intelligence (AI), Machine Learning, Deep Learning, and Neural Networks are broad and occasionally overlapping. Mostly in non-academic sectors, these terms are used as equals, but in the research field, AI and Machine Learning are general definitions that include Deep Learning and Neural Networks.

Neural Networks (or NNs) is a term that commonly refers to the use of Artificial Neural Networks (widely named as ANNs) to imitate human behaviour or a complex task that needs some level of abstraction. But in a real sense, ANNs are just a tool used with other complex technologies that fall in the scope of Machine Learning. The media uses Deep Learning too often, but this term describes more profound and more complex Neural Networks. (Ongsulee, 2017.)

In a broader sense, AI is any kind of intelligence that a computer/machine manifests. Examples of Ais in the past without employing ANNs are search algorithms (Korf, 1996) that run the possibilities in a game to achieve the best outcome possible. These have been used a lot in board games and are the basis of game theory and rational decision-makers. However, Machine Learning is more concrete since it defines a machine or program that learns without being explicitly coded how to perform that objective and how to use the technology for that purpose. Then the term Machine Learning contains ANNs and how they are used (e.g., types of training), along with the investigation and research in ANNs. Generally, Machine Learning is considered a term quite exclusive for the field involving ANNs. However, some processes, such as Markov Chains (Hui, 2020), are also contained within the definition of Machine Learning.

1.2 Advantages of ANNs

The use of these technologies has increased in the past few years. It has begun to develop benefits that seemed impossible in the past, especially to simulate human intelligence for purposes and in ways unimaginable to accomplish until now. One of the most typical examples is the classical problem of number classification, in which the aim is to classify a single handwritten digit. This was broadly studied because of the MNIST database of the National Institute of Standards and Technology (Y. LeCun, Bottou, Bengio, & Haffner, 1998). This example led to the belief that ANNs with enough data could lead to competent performance in an impossible problem to program by a human.

Problems that are not solvable without these technologies share the need for a certain amount of abstraction, which is impossible to describe in a set of discrete rules or conditions. The Neural Network must identify a pattern only achievable with pattern recognition capabilities, and this characteristic present in humans and other species is called "Outline of Thought". We see this concept being used in our daily lives (e.g., voice recognition, marketing, handwriting). (BasicKnoledge101, n.d..)

I decided to use an ANN for recognizing piano notes for many reasons: because of the absence of technologies for music recognition, and Note recognition is another example of a task in which a good design ANN could outperform humans.

2 Machine Learning and Artificial Neural Networks (ANNs)

2.1 Basics of ANNs

Artificial Neural Networks (ANNs) are a series of computing techniques that, in essence, are inspired by biological Neural Networks. The basic structure consists of a group of interconnected elements called neurons. Each neuron transforms the input and passes it to the subsequent neurons trying to accomplish the desired output at the other end of the ANN.

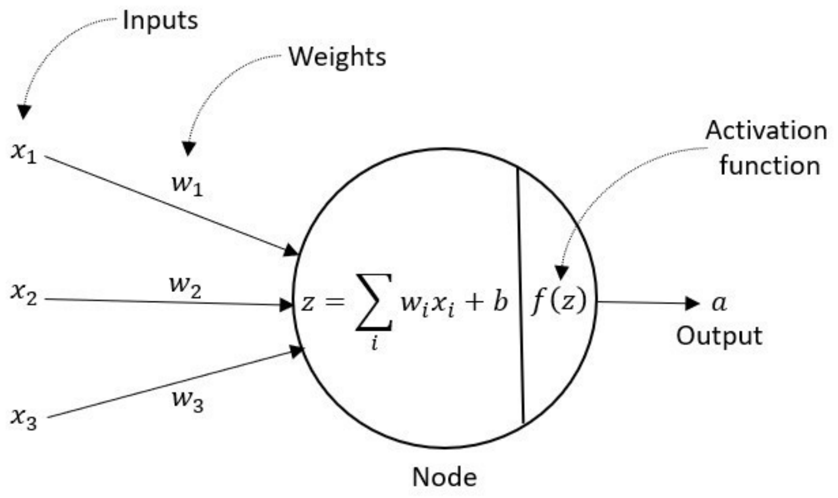


Figure 1. Basic Structure of a Node/Neuron of an Artificial Neural Network (Agarwal, 2020)

A variety of approaches can accomplish the desired behaviour of the Network, but the fundamental way of working is relatively straightforward. First, we will examine a single neuron in Figure 1: how information enters into the system are by the inputs (Xi) and *weights* (Wi), the inputs can be either the main input of the ANN or be derived by another neuron, the weights are the quantity (or importance) of that same input (generally, the connection between weights and neurons is known as a link).

The neuron itself operates by combining the values of its connected neurons with their correspondent weight. To this value, a bias (b) is added, which is a fixed value that can be inputted directly in the neuron. This results in a total (z) that needs to be transformed. The Activation Function does this transformation, the use of this function is mandatory for the ANNs, because in the end, if you don't use this type of function, the ANNs are making a combination of linear regressions. This function brings to the ANNs the nonlinearity that makes them adapt to complex data. (Agatonovic-Kustrin & Beresford, 2000.)

All the neurons form the final ANN, a dense (in most cases) interconnected web of neurons. Figure 2 is a basic structure, but further on, we will see some other examples. The ANN, generally as the basic input will have the raw information that must be evaluated, then it has layers before the output that are called Hidden Layers, which processes the most complicated parts of the input, after that we will have the output. Category problems are one of the most common problems in ANNs, with them the output neurons are used as scores for which category falls the input, but they shall not be confused as a probability because the sum of all neurons will not be 1. There is a wide variety of ANNs, and in more complex examples, the output could be, e.g., connected to another ANN.

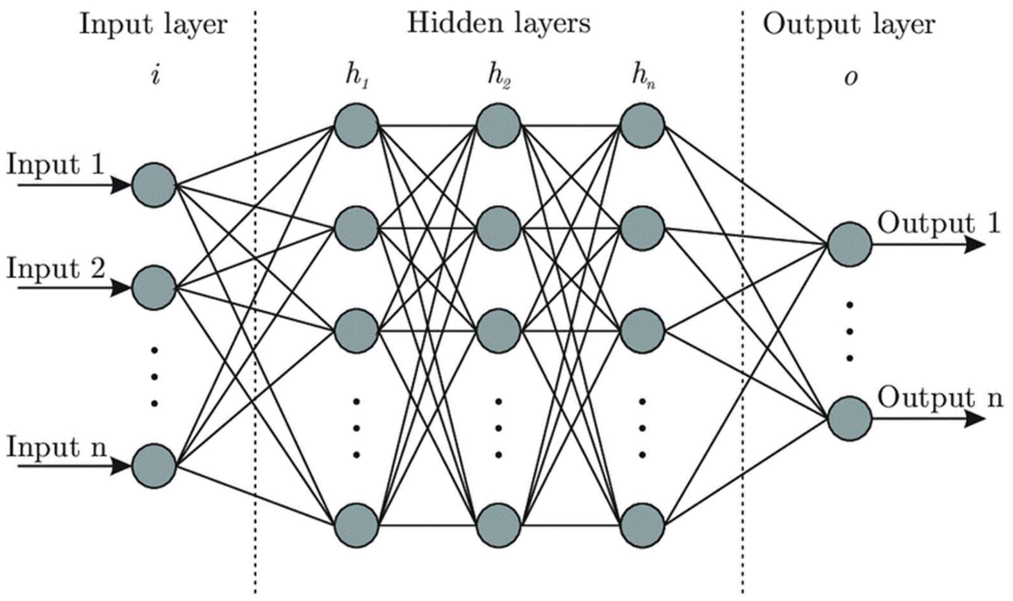


Figure 2. Complete Structure of an Artificial Neural Network (Bre, 2017)

2.2 Structures of ANNs

The internal Neural Network can have different sizes and different ways of organizing the links, and this quality is known as the structure. There are some generalizations, but they must not be misunderstood as if those were the only way of making ANNs since any structure could succeed for a specific purpose. Furthermore, nowadays, there is a lot of research around how new structures can be used for particular tasks. Here we will see the most common and widely known structures. The best way to understand the variety of structural types is to examine the comprehensive collection that has compiled Leijnen and Veen from the Asimov Institute. (Leijnen & Veen, 2020.)

2.2.1 Feedforward ANNs

This architecture is the most straightforward design for ANN, as seen in Figure 3, and the same we saw in the ba-sics, where (as the name says) mainly all the neurons feed the outputs forward. The structure is relatively simple, and it is considered one of the basics for teaching ANNs. Feedforward ANNs are useful in simple problems and could be suitable in more complex problems, with the downside of needing many layers and possibly leading to never-ending training.

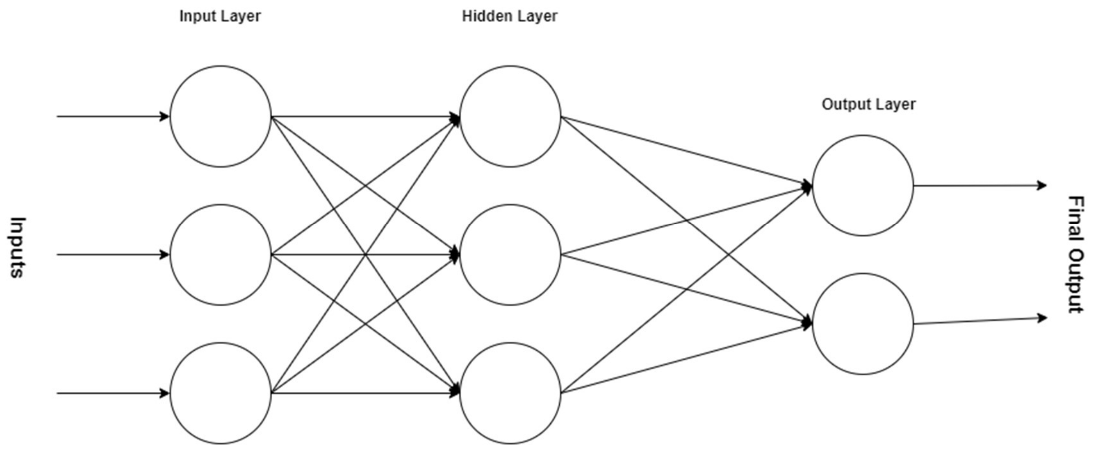


Figure 3. Structure of a Feedforward ANN

For educational purposes, it is worth mentioning that the simplest ANN is commonly called "perceptron", a feedforward ANN with just one layer. The perceptron is a binary classifier, where the output layer contains the probabilities of the input falling in one category or the other, as we see in the example in Figure 4. It is easy to understand how the neurons determine how much of a value is essential or if a value is relevant just in one layer. (Nor-iega, 2005.)

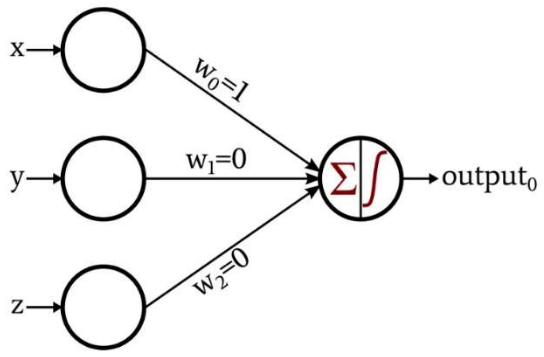


Figure 4. Structure of a Perceptron (Keim, 2019).

2.2.2 Recurrent

Recurrent Neural Networks (RNNs), like the example in Figure 5, are a broad term referring to ANNs that include loops in the structures of the ANN, which means that the information previously seen is being considered. This type of Neural Network technique has practical uses in society, mainly in problems in which the previous information is useful, e.g., audio recognition or image characterization. This type of technique is thought of as a similar equivalent to short memory, but they are task-dependent. The primary way of dealing with input information is that the input has to depend on the time. The most common structure has the RNN input separated by splices of time, and each iteration feeds the RNN the next slice of time. In a simple example, you can think this as if you already had seen some object in a position, the likelihood of seeing it again in that position is higher than just before you recognized it. Previous information greatly impacted the state of the output. (Biswal, 2021; Elman, 1990.)

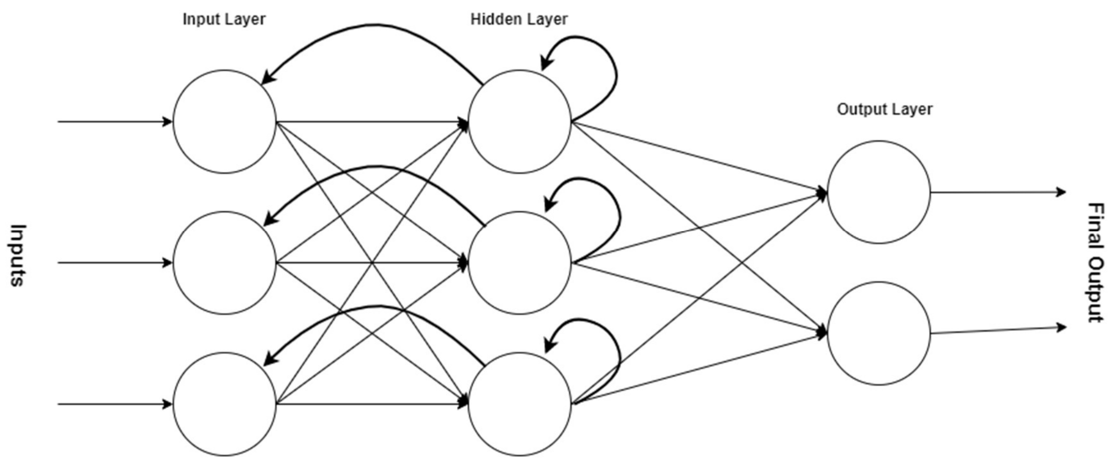


Figure 5. Structure of a Recurrent ANN

Another way of looking at how RNNs work is like a series of states or copies of the Network in which each time the Network is inputted new data (maybe in real-time) and at the same time previous data from the last state. This may sound complicated, but it is easy to understand seeing Figure 6. The main problem of these Networks is that long-term memory is typically lost, and only near past information is being influenced in the present state of the Network.

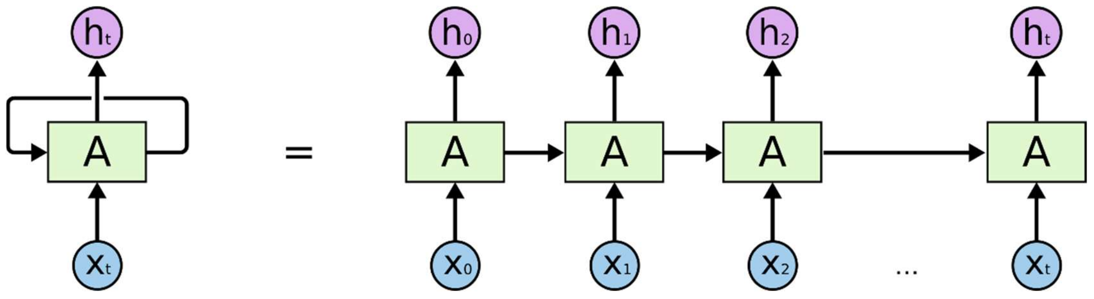


Figure 6. Example of an RNN expanded (Olah, 2015)

2.2.3 Autoencoders

This type of ANNs is worth mentioning because they reflect how this technique can be pretty surprising or even unexpected considering the original purpose. As seen in Figure 7, the functioning of Autoencoders is divided in two: one part of the Neural Network encodes while the other decodes, with some neurons in between. The purpose of encoding is to get rid of useless information, often called "noise", then passing the essential information of the input to the centre notes. This critical information gets propagated through the decoding part, which tries to reconstruct the input using but without the unwanted information and with the same general shape.

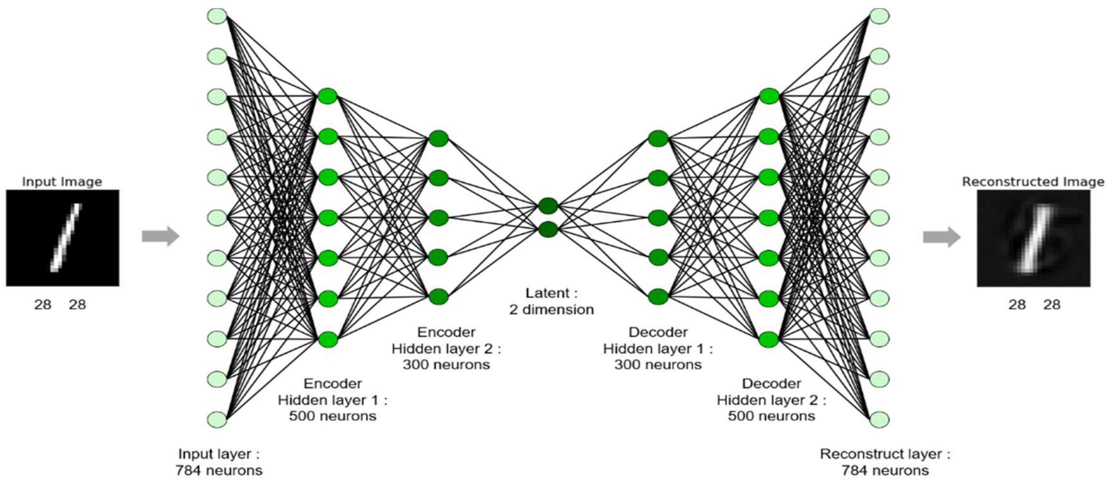

Input Image
28 28
Input layer : 784 neurons
Encoder Hidden layer 1 : 500 neurons
Encoder Hidden layer 2 : 300 neurons
Latent : 2 dimension
Decoder Hidden layer 1 : 300 neurons
Decoder Hidden layer 2 : 500 neurons
Reconstruct layer : 784 neurons
Reconstructed Image
28 28

Figure 7. Structure of an Autoencoder ANN (Box, 2019).

This technique was thought to have some very concrete uses, like some mathematical purposes or some utilities for computer databases. One interesting use is for anomaly detection and image compression. The ANN extracts the essential and tries to build an equivalent, if the input contains some anomaly that makes it unsuitable, even if it is not recognizable by a human, it will fail in creating a suitable reproduction of the input.

2.2.4 Generative Adversarial Networks (GANs)

The Generative Adversarial NNs, called with the acronym GAN, have gained popularity in research and the media. This paradigm of ANNs is not actually one ANN, they are two competing against each other, and they can have any structure. One of them tries to generate some data (Generator), the other (Discriminator) tries to predict whether the data has been generated by the Generator or comes from natural samples. This behaviour can be seen in Figure 8, with this structure both fight for accuracy: the Discriminator trains to determine whether the Generator is producing well enough reproductions not to be distinguished, and tunes if the Discriminator is predicting good enough. This results in one Network that creates really accurate fake samples and another Network that can recognize real examples accurately. This architecture has received a lot of attention for creating fake images, e.g., NVIDIA (Karras, Laine, & Aila, 2019) released a GAN called StyleGAN that could create accurate people's faces, which got a lot of popularity by the website "*thisperson-doesnotexist.com".* (Goodfellow et al., 2014; Leijnen & Veen, 2020.)

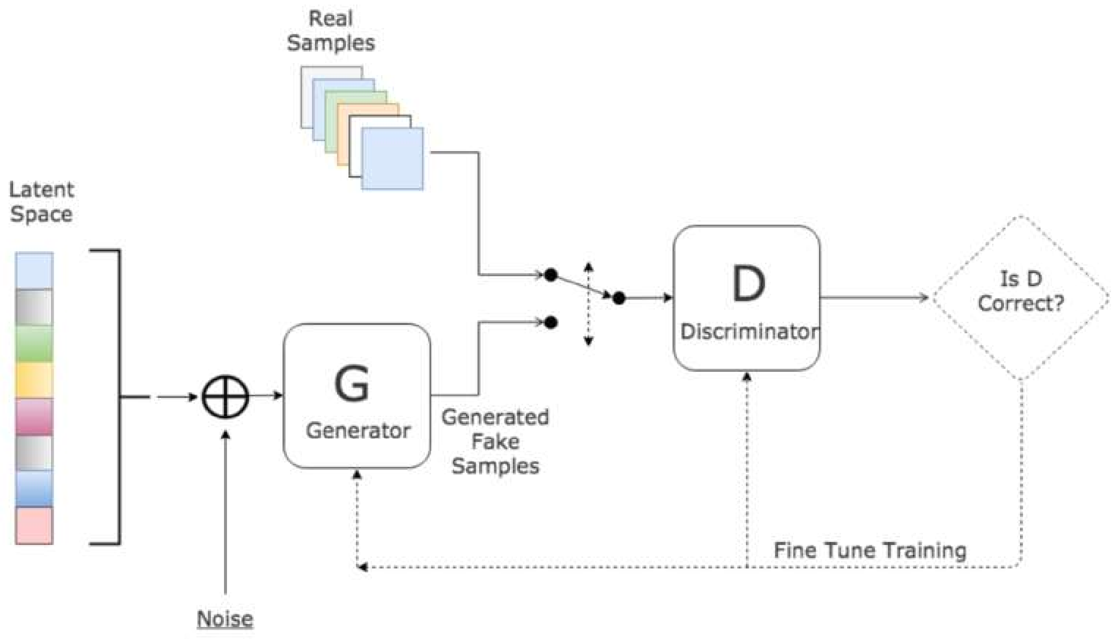


Figure 8. Structure and work of a Generative Adversarial Network (Gharakhanian, 2016)

2.2.5 Convolutional Neural Network (CNN)

CNNs are the most complex that we will discuss, but we must mention them because their use in the past years has become quite extensive, and they provide impressive utility in image analysis.

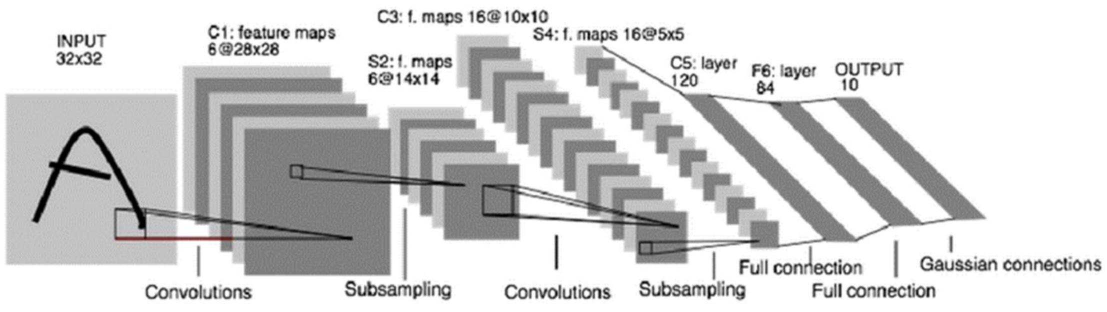


Figure 9. Structure of a Convolutional ANN (Lowndes, 2015)

As seen in Figure 9, this ANN works differently by having parallel independent layers. These parallel layers are called "feature maps" or filters. These layers produce a transformation of the input in which some features are removed or highlighted depending on the filter. The main way these ANNs work is by establishing a correlation between the location of the image's pixels (for example) and the later operations inside the Network. In the past examples, we have seen Networks in which all neurons of the same layer were linked to all neurons in the next layer, but in this case, just some links are established and, therefore the neurons that are closer together get linked to one neuron, e.g., the pixels being near each other. This results in compressed information because as information is lost layer after layer, they behave like autoencoders, where useless information is not used in deeper layers. Finally, information gets extracted from subsets of the image and then from even smaller subsets in the next layers to a final classification with all fully interconnected layers for the conclusion or classification that the ANN tries to pursue. (Albawi, Mohammed, & Al-Zawi, 2017; NVIDIA, 2018.)

From this point, we will describe more technical aspects, so for the sake of clarity, every time we refer to a convolutional layer, we refer to all the parallel filters alongside their unique operations.

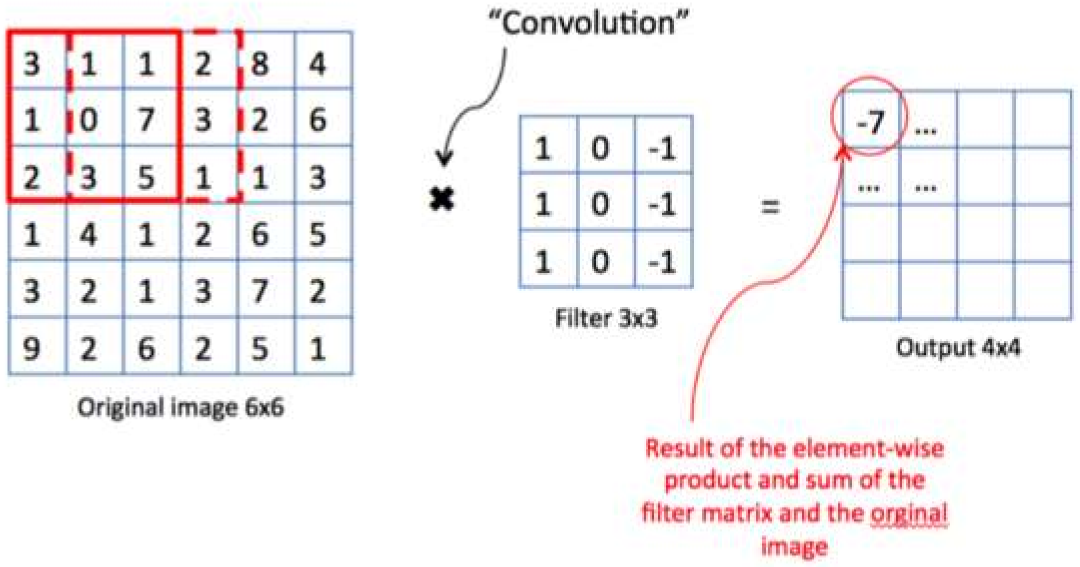


Figure 10. Example of a Kernel Matrix/Filter in a Convolutional ANN (Cavaioni, 2018)

To comprehend how this ANN works, we should look into how the actual convolutional layers are defined. Most of the time, it seems that there are just 2D layers with a specified size, but it is pretty more complex since it is a 3D layer. It is important to note that the same convolutional layer (same parameters) with different inputs will have different width and height at the output. In the end, the layer just makes a scaled version of the last layer relating the near contents. This is done using a Kernel Matrix (also called filter) instead of establishing many links between input and the neurons. Each filter (layer) establishes its Kernel Matrix with an activation function and moves it around with all the input values, generating the output. We can see an example of this operation in Figure 10. (Stanford University, 2021.)

All we have seen lead to see that all that makes the actual convolutional layer are mainly the parameters of the Kernel Matrix, here we will see an explanation on all parameters.

- Depth: states the number of filters or parallel layers present.
- Filter/Field/Kernel size: the size of the Kernel Matrix, typically states the two dimensions of the Kernel Matrix as one equal number but may not be square.
- Zero Padding: surrounding the input borders are present the number of zeros stated by this parameter.
- Stride: the number of positions that shift the Kernel Matrix as it moves with the input values, is typically left to 1. But in cases in which overlapping values may interfere will be set to a higher number

One way of understanding how these parallel layers (filters) work is knowing that each one of the parallel layers is extracting features. The first filters just extract very concrete characteristics, and the subsequent layers mix that information to more general traits. In Figure 11, we see precisely the transmission of these parallel layers. The first one is nothing understandable but the next combine to a broader understanding. (Albawi et al., 2017; Zeiler & Fergus, 2014.)

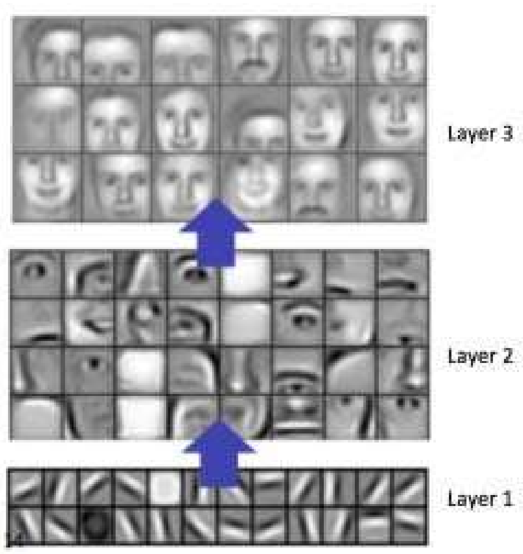


Figure 11. Feature selection done by a Convolutional ANN (Albawi et al., 2017)

2.3 Activation Functions

Activation functions are essential for the ANNs because they bring nonlinearity. Linear functions as activation functions will just try to discriminate using straight lines (like linear regression). In real applications, ANNs use nonlinear functions to distinguish data using any shape. The figures below portray the same problem with one hidden layer, but in Figure 12, an ANN tries to discriminate input data using linear functions, and in Figure 13 is trying to use nonlinear functions. This indeed is the most appealing behaviour of ANNs, the ability to adapt according to a system, a system that could be incredibly complex. (Sagar Sharma, 2017; Shikhar Sharma, 2019.)

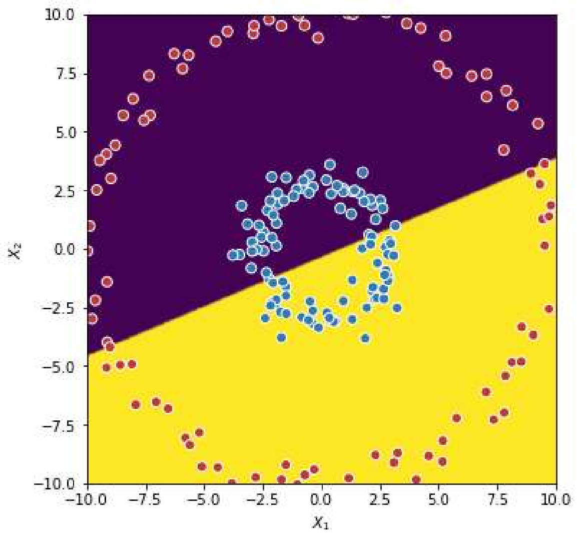


Figure 12. Example of using a linear function inside an ANN for resolving a nonlinear problem (Shikhar Sharma, 2019)

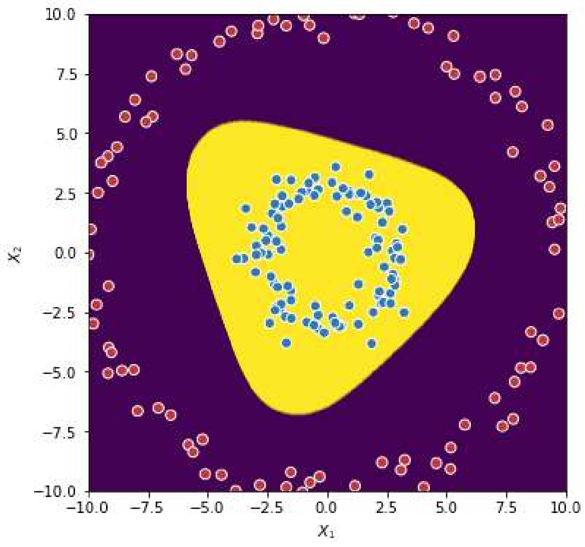
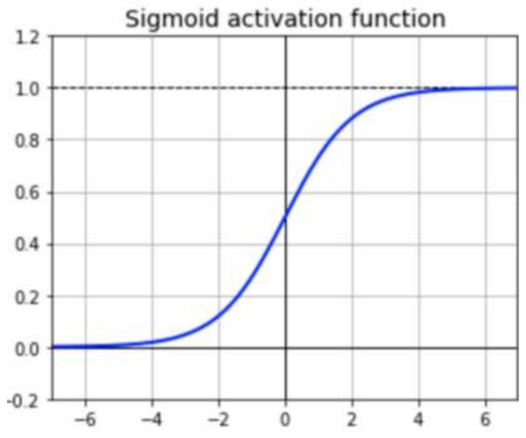


Figure 13. Example of using a nonlinear function in an ANN for resolving a nonlinear problem successfully (Shikhar Sharma, 2019)

Until now, we have seen quite simple operations for the basics, but the code implementation needs a lot of complexity, mainly in making the Network evolve depending on the outcome during training. For this purpose, the techniques called Optimization Algorithms or Backpropagation Optimization are used, and we will see them later, but for using them the activation functions have to be differentiable as all the functions explained in the next section. (Jordan, 2017; Sagar Sharma, 2017.)

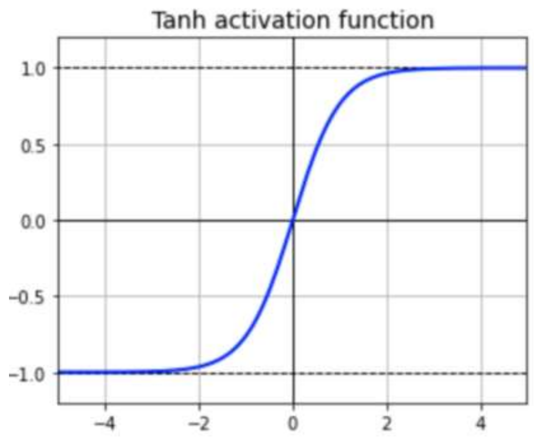
2.3.1 List of the Activation Functions Most Used

The Sigmoid Function was the first nonlinear function used in modern ANN. It acts similar to a predecessor called the Step Function, but this function just outputted a cero if it was less than zero and one otherwise. As seen in Figure 14, the Sigmoid Function replaced the Step Function by having that s-shape between extremes. Its primary use is in classification problems, and one of its advantages is that the values will not increase in deeper neurons because the output is bounded. (A. Sharma, 2017; Sagar Sharma, 2017.)



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Figure 14. Sigmoid Activation Function and its Graph (Chen, 2021)

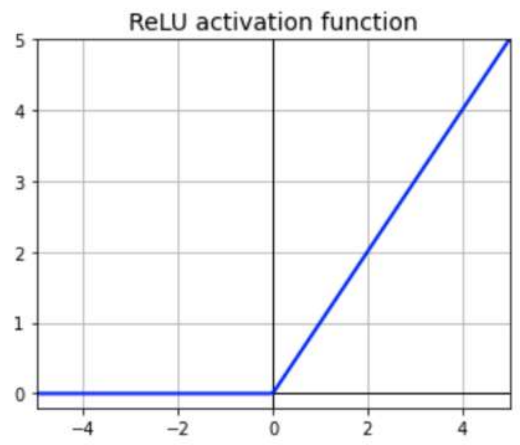


$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Figure 15. Hyperbolic Tangent (Tanh) Function and its Graph (Chen, 2021)

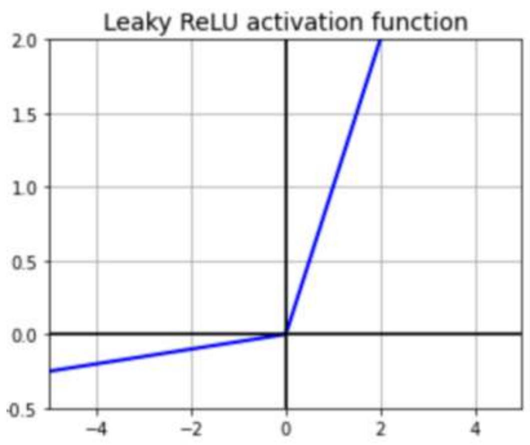
The Hyperbolic Tangent Function (or Tanh) is similar to the Sigmoid Function, but it came out as a better version. In Figure 15, we see the main advantage, it is centred in zero, making layers centred around 0, and the steeper Gradient makes the derivatives steeper helping the optimization. The Tanh is seen as a scaled better version of the Sigmoid Function, and it is one of the most used nowadays, mainly in classification problems. (Chen, 2021; Stanford University, 2021.)

The biggest problem that the previous two functions have is called the "Vanishing Gradients". This means that the output cannot be distinguished in extreme values (>4 or <4) because of near values in the Y-axis. The differences that the ANN is experiencing are literally "vanishing", which makes either the ANN not learning or just evolving drastically slow. This behaviour is usually caused by the limit of the floating-point numbers used in computers today, which cannot have more than a finite number of decimals, and after that, every value is the same. (Brownlee, 2021; A. Sharma, 2017.)



$$\begin{cases} 0 & if \ \ z \leq 0 \\ z & if \ \ z > 0 \end{cases}$$

Figure 16. Rectified Linear Unit (ReLU) Function and its Graph (Chen, 2021)



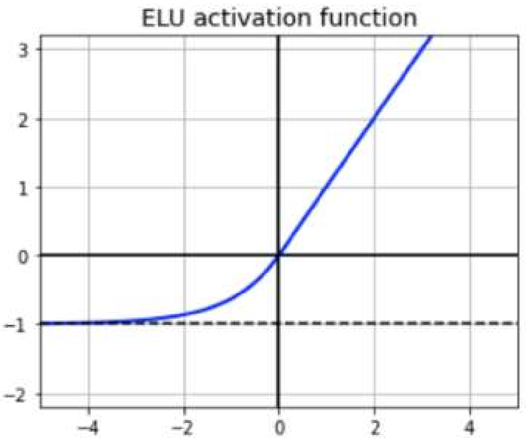
$$\begin{cases} 0.01z & if \ z < 0 \\ z & if \ z \geq 0 \end{cases}$$

Figure 17. Leaky ReLU Function and its Graph (Chen, 2021)

The Rectified Linear Unit Function (or ReLU) is one of the most used functions in ANNs because of its versatility, and in a lot of cases, one of its variations is used. As seen in Figure 16, the function seems pretty linear, but it is nonlinear, and its combinations will also be nonlinear. Despite its appearance, the combinations of ReLU could approximate any function. The function acts as a switch killing the neuron before 0, and after that, it can output a variety of linear values depending on the input and state of the ANN. But these low values have the problem of causing neurons to stop forever, and in cases, a considerable part of the ANN turns off, this is called the "death of neurons". Despite that last inconvenience, one of the reasons for using ReLU function is that it is cheaper computationally, not using exponents makes it simpler for processors. (A. Sharma, 2017; Stanford University, 2021.)

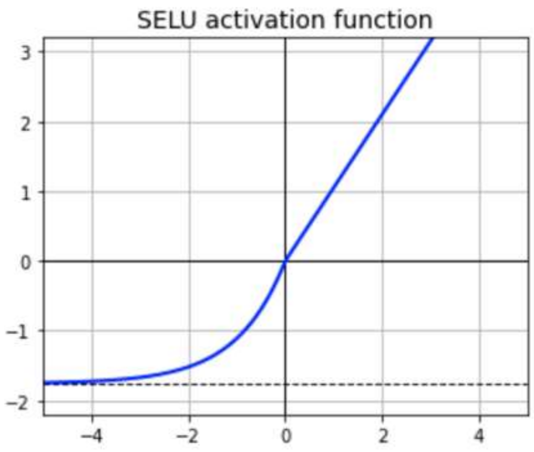
For the problems that experienced ReLU, other variations came out. One common variation was the Leaky ReLU. The modification present in contrast with ReLu is easy to spot in Figure 17, it has the purpose of stopping neurons from dying but maintaining its simplicity (computationally speaking). Some cases seem to highly benefit from this function (Xu et al., 2015), but it's highly debatable if it can be used widely or if it should be used in cases in which ReLU is not working because of the "death" of neurons. (Stanford University, 2021; Versloot, 2019.)

Other possibilities of ReLU differ with different values of the product in the leftmost part, to change the slope in Leaky ReLU. Even establishing a random value for the slope, which is called Randomized Leaky Rectified Linear Unit or RReLU. In some papers, it seems that it can be beneficial, but its utility is in doubt, similar to Leaky ReLU. Nevertheless, both of them need more investigation. (Xu et al., 2015.)



$$\begin{cases} \alpha(e^x - 1) & if\ z \leq 0 \\ z & if\ z \geq 0 \end{cases}$$

Figure 18. Exponential Linear Unit (ELU) Function and its Graph (Chen, 2021)



$$\lambda \begin{cases} \alpha(e^x - 1) & if\ z \leq 0 \\ z & if\ z \geq 0 \end{cases}$$
$$where\ \lambda = 1.0507$$
$$\alpha = 1.67326$$

Figure 19. Scale Exponential Linear Unit (SELU) Function and its Graph (Chen, 2021)

The Exponential Linear Unit Function (ELU) is another version for fixing the problem with dead neurons, but while the success of the Leaky ReLU is disputed, with ELU the success is evident. This function, with the tests performed by Clevert et al. with α=1 as seen in Figure 18, speeds up learning in comparison to ReLU and Leaky ReLU, and an outstanding result in CIFAR-100, which is a standard dataset of images used for benchmarking ANNs (Krizhevsky & Hinton, 2009). Although in most tests the ANN get the same results in fewer iterations, the training takes longer because the function is more expensive in terms of computation. As fewer iterations do not result in better times, it is not used much, but with faster exponential calculations it could be used more in the future. (Clevert, Unterthiner, & Hochreiter, 2015.)

One of the most modern ones to be touched in this paper is the Scale Exponential Linear Unit (SELU) function, proposed by Günter Klambauer et al. (Klambauer, Unterthiner, Mayr, & Hochreiter, 2017). This function is seen as a similar version of the ELU function, as seen in Figure 19, but the properties made this function promising. In the original paper using this function makes each hidden layer self-normalize, this meant that each layer would tend to have a mean of 0 and a standard deviation of 1, which makes the ANN works more evenly. It resolves the problem of vanishing Gradient and produces more stable Learning. This function seems to outperform most of the activation functions, but it has conditions that most of the activation functions do not have, it needs to be initialized with LeCun normal initialization (Y. A. LeCun et al., 2012). (Sagar Sharma, 2017.)

$$\sigma(\vec{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1 ,\ldots, K \text{ and } \vec{z} = (z_1 ,\ldots, z_k) \in \mathbb{R}^K$$

Equation 1. Softmax Function

It is worth mentioning a function that is considered an activation function most of the time, but its use is quite similar. Equation 1 is called the Softmax function, often also called the Softargmax function in other fields. In ANNs, this function takes the values outputted in the Network's last layer and converts them into a set of probabilities. The calculated values are bounded to the interval (0,1), and the sum of all of them will be 1. We can see an example in Figure 20 and how it converts any value to a normalized probability. This is why its use is entirely exclusive to classification problems when the classifying classes are mutually exclusive. (Wood, 2019.)

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 5 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} \sigma(\vec{z})_1 \\ \sigma(\vec{z})_2 \\ \sigma(\vec{z})_3 \end{bmatrix} = \begin{bmatrix} 0.9523 \\ 0.0474 \\ 0.0003 \end{bmatrix}$$

Figure 20. Example of use of the Softmax Function

2.3.2 Relation with Bias Value

The bias parameter most of the time is left and not explained, but it is one of the most manageable parts to understand. This value has a lot of utility because, in a way, it makes a threshold for the activation function. It takes the activation function and shifts to a direction making its values more/less achievable at a certain point. In Figure 21, the sigmoid function is shifted to both directions depending on the bias value. The model with the bias was not introduced with the first ideas of the perceptron in the fifties, but its usability makes it nowadays common practice. (Kohl, 2010; Zilouchian, 2001.)

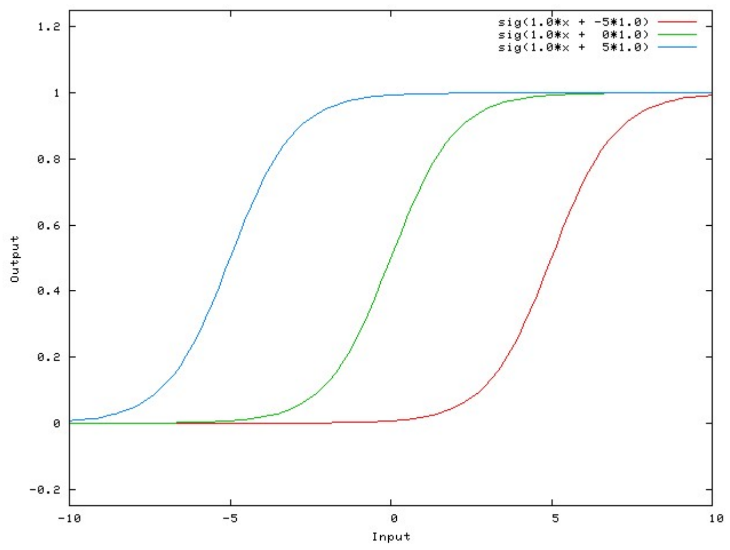


Figure 21. How the Bias value changes the Activation Function (in Sigmoid Function) (Kohl, 2010)

2.4 Training ANNs

One of the most intriguing terms that have described this field and the making of ANNs is "black box", we can see, e.g., a publication of Nature journal (Castelvecchi, 2016). This term is not commonly used in scientific papers, but widely defines that when we trained these types of ANNs, we do not have a way of knowing how information is treated or how the information leads more towards one decision or another, we just input some numbers, and some result comes from the output. This concept is considered a drawback by many because the process cannot be studied, the outcome that we have is a learning machine that has accomplished a task that a human cannot code. Nevertheless, as many recall this field to be unsuitable for the scientific community, most of the advancements made in this field have been towards better practices in training ANN and ways of making them understand more complex data or reproduce it. (Azodi, Tang, & Shiu, 2020; El Naqa, Li, & Murphy, 2015.)

Usually, this topic should fall in the Machine Learning field because this field studies these types of questions, but we will focus just on ANN. Until now, we described simple operations that happen from neuron to neuron of an ANN, but the amount of complexity that the field has poured into training is extensive. After all, this is the nexus of having the final ANN acting as a real AI. The improvement in more complex training techniques has increased the urge for large databases (Siddharth Sharma et al., 2020), like the Sports1M (Karpathy et al., 2014) and competitive software like TensorFlow. This has led to further investigation for better techniques in more complex scenarios. (Samek, Wiegand, & Müller, 2017.)

2.4.1 Supervised Learning

Supervised Learning is a common name for using labelled data for contrasting if the ANN is correctly guessing from the input data. It is called "Supervised" because we accurately know if it is guessing each input correctly. These methods are best suited in problems where a large dataset was generated, but there may not be an available database in complex problems. Therefore creating one can be pretty time-consuming. (NVIDIA Blog, 2018.)

There are two main types of problems in Supervised Learning:

- Classification: One of the most common problems inside ANNs for its easy implementation. The ANN must predict if the input data fits in a particular category, and then it can measure how much it has failed. One example of such can be an ANN that can identify the animal in the image.
- Regression: In this problem, the ANN learns the relationship between dependent and independent variables. It is different compared to classification because it gets inputted continuous data. The best way to understand this problem is when the ANN gets asked *given a particular x value, what's the expected value of the y variable?* (NVIDIA Blog, 2018).

In the practical case, I have chosen this type of Learning because I could generate an accurate dataset, and the problem was indeed a classification problem.

2.4.2 Unsupervised Learning

Unsupervised Learning is the technique of training ANNs by using unlabeled datasets. The idea of training without guidance can seem complex, and indeed the predictions used by this type of Learning usually have less accuracy than in Supervised Learning. But this problem is beneficial in real-life applications, e.g., buyers who tend to buy two types of products. (Delua, 2021.)

There are two main types of problems in Supervised Learning:

- Clustering: The ANN takes a large quantity of data and groups them based on similarities. One example is trying to differentiate similar bird species without telling the ANN the actual species but just looking at the differences. This is easier to understand by examining Figure 22. This technique is the most common in Unsupervised Learning.
- Association: This type of training is done to make relationships between subsets of the Dataset, so it is used to know the most common associations between specific data attributes. Association is universally used in trending prediction, e.g., *Fill an online shopping cart with diapers, applesauce and sippy cups and the site just may recommend that you add a bib and a baby monitor to your order* (NVIDIA Blog, 2018).

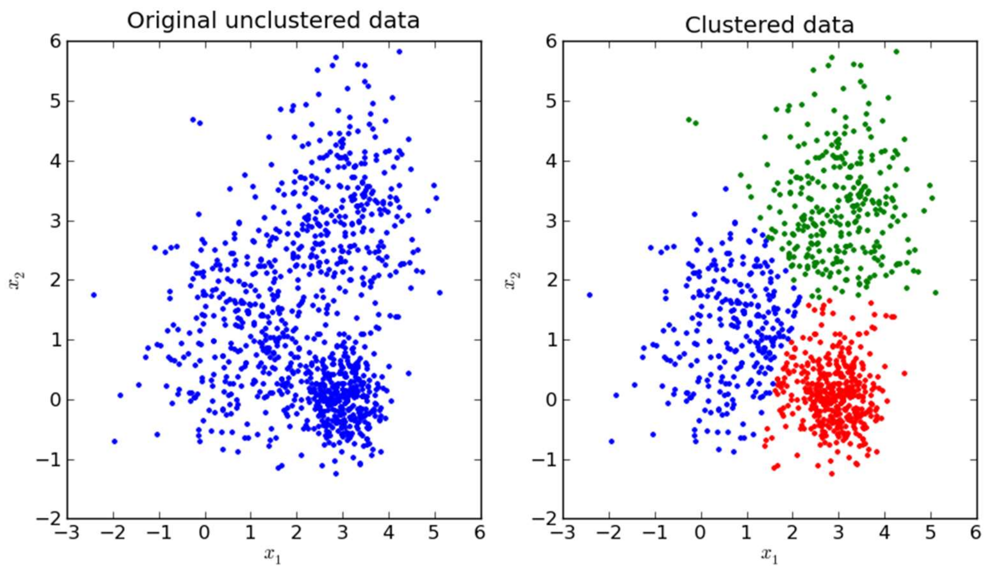


Figure 22. Example of Clustered and Unclustered Data (Ruiz Ruiz, 2018)

2.4.3 Stochastic Gradient Descent and Backpropagation

Until now, we have seen everything of what it is the ANN, but we will see how its training happens.

Before talking about Stochastic Gradient Descend, we must speak about Backpropagation (short of *backward propagation of errors*). They are highly related because for using Gradient Descend, you first apply Backpropagation. Backpropagation is an algorithm for calculating the Gradient of the error (also called Loss) concerning the ANN weights. This means that it estimates how changing the weights will change the Loss to better/worse results. But we do not know the optimal value, just the approximate direction from the actual state. The Gradient is calculated first in the last layers and finishes at the first layer. This flow of calculations allows for efficient computations, and it propagates the error that we certainly know at the last layers towards the first layers. Here we have the reason why the Activations Functions needed a derivative, the Gradient is calculated with partial derivatives. (Brilliant, n.d..)

Backpropagation is a vast topic, and if you want to learn more about how it works (in practice), you should check the course of 3Blue1Brown, a great content creator (3Blue1Brown, 2017).

In the practical case, I used the Stochastic Gradient Descent (SGD), a derivation of the Gradient Descend. The Gradient Descent is one of the bases for most algorithms used in training ANNs. It is the function that calculates how the weights should be changed to get near the optimal behaviour for the ANN, basically the point in which the Gradient is 0, in Figure 23 we can see some of those points which in every direction the function has a higher Loss. But normal Gradient Descent just goes in the direction of the slope by calculating an average with many results. As a problem, this can make it reach a point of 0 gradient without being the actual minimum (local minimum). (Azizan & Hassibi, 2018; V Srinivasan, 2019.)

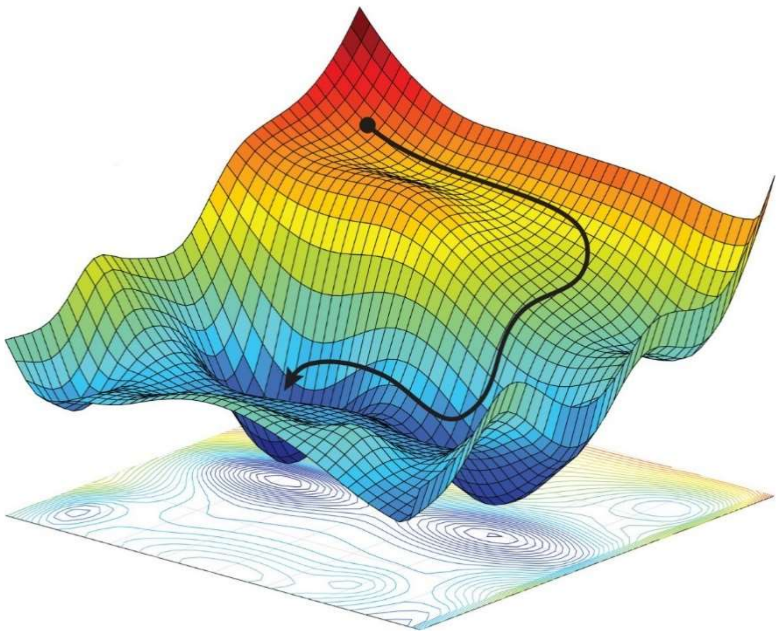


Figure 23. Example of how a Gradient Descend gets to the most optimal value (Azizan & Hassibi, 2018)

The Stochastic Gradient Descend comes to resolve the previously seen problems. In this case, the SGD, instead of developing an average of many results, takes one random individual result to calculate the following weight change. This change makes it faster in terms of computation and resolves the problem of local minimums. As a downside, it is more unstable, but these "random" changes approach the optimal solution with enough data. The SGD is one of the most used learning algorithms, it is one of the more robust and faster. (V Srinivasan, 2019.)

3 Technologies for Audio Classification

3.1 Introduction

Audio Classification is considered one of the most complicated problems compared to other classification problems inside this field. It is easy to collect raw audio and even filter out all the useless noise in the background but even after that, obtaining valuable information is an uphill task. However, newer techniques have led to faster and easier implementations for solving these problems. Much of the actual methods use a time distributed approach, which means that splits all the time into evaluable audios with the same length. Most of the time, these split-audios are the input for an ANN. (Lu, Zhang, & Nayak, 2020.)

3.2 Use of Artificial Neural Networks

3.2.1 Convolutional

The use of ANN has changed the way we see audio classification/recognition. One of the most common techniques for approaching these problems is CNN, which is done in most cases because the problem is bound to single files, like distinguishing bird species (Satyam Raj, 2020).

The way of working can be divided into two: waveform and spectrogram work. In the practical case exposed later, I face this problem using spectrograms: the audio file is converted to a spectrogram. A spectrogram at the end is all the frequencies plotted through time by its intensity, as seen in Figure 24. This makes the input of the CNN in 3D form, and the use of conversion in preprocessing is needed. In contrast, pure waveforms are just inputting the 2D raw audio information to the CNN. This raw input in some real-time processing is better, but in many cases, the use of spectrograms outperforms or is more accessible to implement than raw audio. (Lu et al., 2020.)

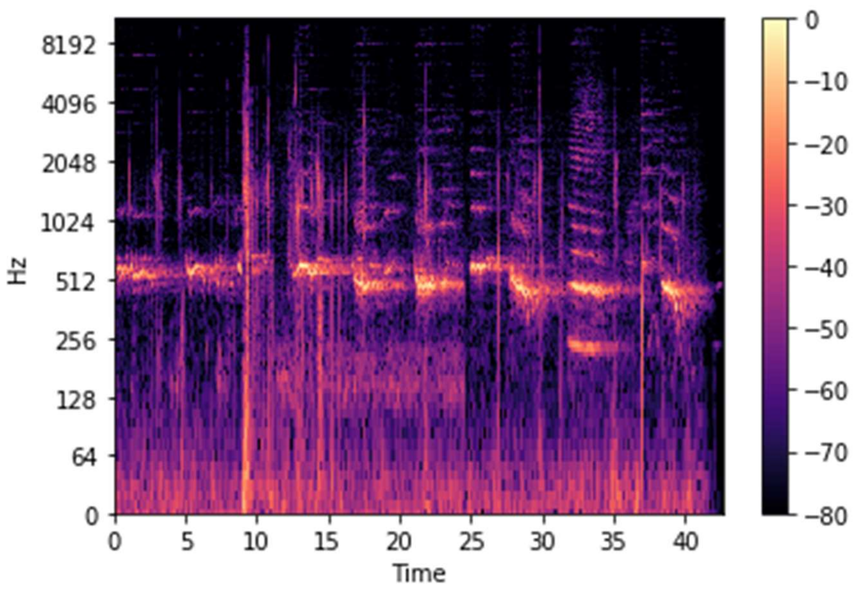


Figure 24. Example of a spectrogram (Gartzman, 2020)

3.2.2 Recurrent as Long Short-Term Memory (LSTM) Networks

Using a Recurrent Neural Network can be an advantage in contrast to CNNs if the data that will be used as input is time-dependent. Mainly used in cases where the audio may not be complete in one input of the ANN, and then it is made as audio that will be arriving as it is produced (like a microphone). Similar structures to the seen in Figure 5 may work for near time dependence, but if there is a need for long-term dependence, these Networks could be inadequate to train because as the dependencies are longer, the Gradient Descend for training becomes increasingly inefficient. (Bengio, Simard, & Frasconi, 1994.)

For resolving long-term dependencies, the LSTMs Networks were created. This type of Network was explicitly designed to maintain information for an extended period of time. The typical structure of the LSTM networks can be seen in Figure 25. It may seem intimidating, but it is not that complicated when you understand all the operations, as this Network is based on parallel Softmax (σ) and Tanh layers. It divides into two parts, in which resides its utility, the upmost long connection and the low part with Layers. The part that maintains the information for long periods is the uppermost part and is called the LSTM Memory. This line of information runs almost unchanged to the next state, it may remove information by the pointwise multiplication, but indeed, after training it just adds to this line the essential information and rarely gets removed, just overwritten. (Olah, 2015.)

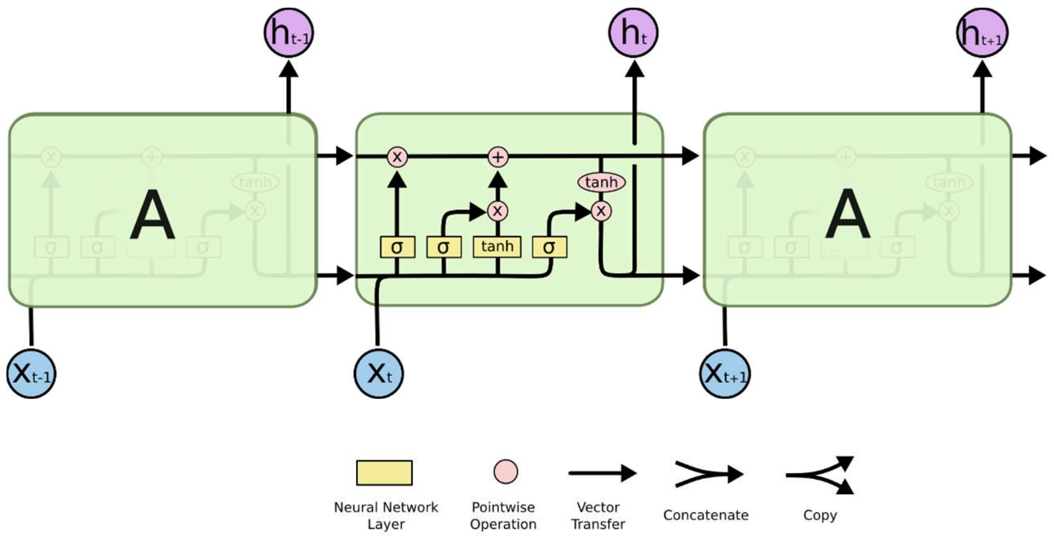


Figure 25. Example of a standard structure of an LSTM (Olah, 2015)

The LSTM technique comes in good hands in files with unknown time, then splitting them for the Network will be helpful. LSTM has been used as a better and more professional version of RNN. Its use in Audio is quite extensive, e.g., Music Genre Classification. (Premtibadiya, 2020.)

4 Case: Piano Note Classification Aided by Convolutional Neural Networks

4.1 Introduction

Most musicians can identify some notes, and even some of them can identify chords. This musicians' ability is commonly called Absolute Pitch or Perfect Pitch, because they can identify music notes without a reference. This ability is quite rare, and it seems 4% of music students possess this ability (Carden & Cline, 2019). For me, it was outstanding that these musicians could identify the exact pitch of everyday sounds like alarms or car horns.

This type of ability made me believe that an ANN could also identify sounds as different patterns. The clear pattern that a note made in a spectrogram, as in Figure 26, and the absence of note recognition software in an environment of high non-similar background sound, made me go for making an ANN that could identify notes.

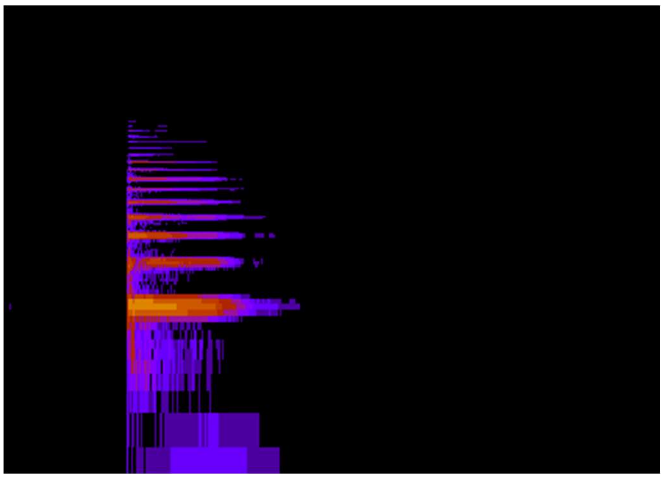


Figure 26. Spectrogram of a C Piano Note in one of the audios generated

Before continuing into the practical case, it is worth mentioning that I made all decisions by myself. It was a complex task because I already knew the theory about this field, but it was the first time that I used any of the libraries seen below.

During this practical case, it is evident that one of the most important tasks is creating the Dataset and that is why 90% of the time has been spent preparing a competent dataset. In the training of ANN, the training data is essential, it needs a big enough Dataset with data independence; this is an underrated topic in my opinion. Extensive research revolves around the importance of robust and reliable training data for the successful implementation of an ANN.

In order to generate the dataset used in this work, I focused just on the seven central notes of the piano, also called C4 to B5. This focus on the central notes is made as a compromise for simplicity in showing that these techniques can function with them, and indeed working with the rest of the notes, would only require a larger dataset and computing time. In my opinion, the reason for the good results I achieved is the quality of the Dataset.

For understanding the time estimations mentioned in the case, I have added a summary of the system's main characteristics, as the Table 1. The CPU and the Main Hard Drive are the ones that are present in all the operations, but the GPU and the memory are predominantly present during the training of the CNN because Tensorflow supports CUDA execution.

CPU	GPU	Main Hard Drive	Memory
I7-6700k	Nvidia GTX 970	Intenso SATAIII 512GB	Kingston HyperX DDR4 2400 C12 2x8GB

Table 1. Characteristics of the Computer during all execution

4.2 Creation of the Dataset

One of the first decisions that must be made is how the information of the notes will be stored in the audio files. For that, I have chosen all audio files to be Waveform Audio Format (.wav) files of seven seconds that contain a note (or simultaneous notes in some cases) with a random duration of [50,1500] milliseconds and a random beginning position in the audio, taking into account that a note could not be cut at the end/beginning of the audio. The reason for using .wav files instead of other more ordinary formats as mp3 is due to the fact that in this format the audio´s information is not compressed, and this is desired for the training of ANN.

After many tries and dealing with RAM limitations and training with different amounts, the standard quantity of files that work in all cases is 6500. This is the maximum quantity of files that my computer with the GPU can process simultaneously, with 11 GB of RAM allocated.

4.2.1 DryWetMidi Library

The other important aspect is how these audio files will be created. For this first step, I have chosen to create first files that comply with the Musical Instrument Digital Interface (or MIDI) Standard. These files do not store audio, they just keep the information of where the note is located according to the tempo of the music and what their duration will be. The reason for the creation of these files is because it is an international technological standard, and for the fast generation of the Dataset it is an appropriate step. Most other libraries (as the one used later) use this format.

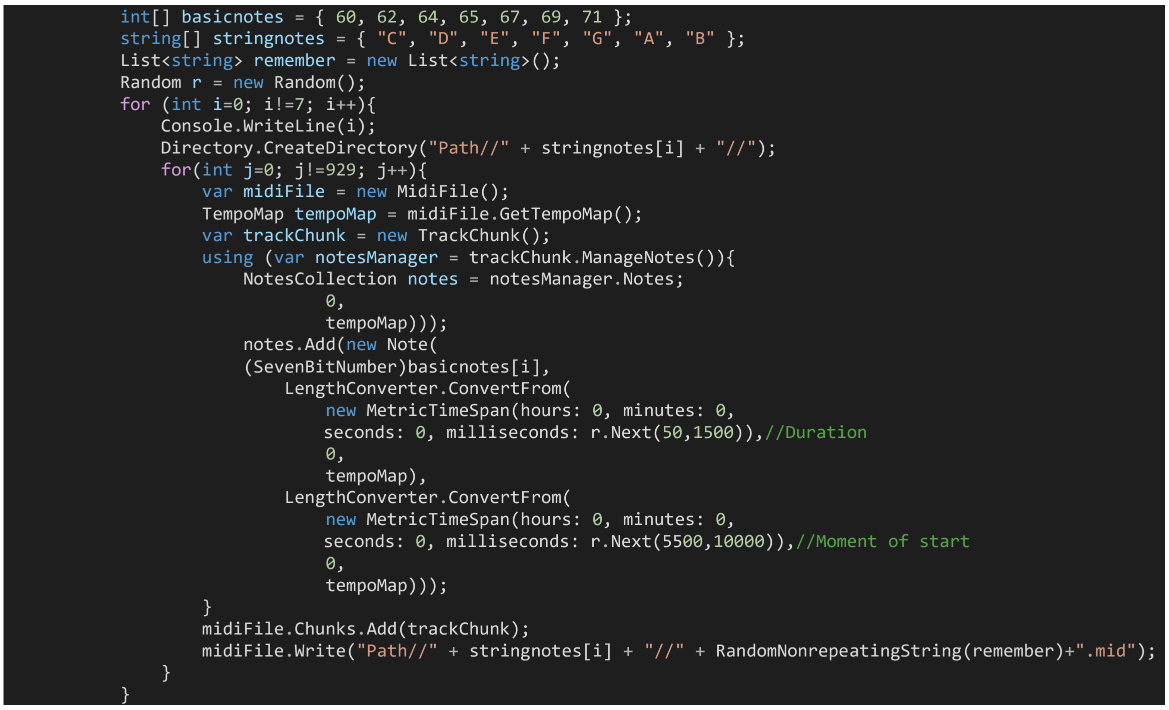
DRYWETMIDI

Figure 27. Logo of the DryWetMidi Library (Dobroselsky, 2017/2021)

The library that I felt helpful for creating MIDI files, after searching many in different programming languages, was DryWetMidi in C# programming language. It is a complex library by Maxim Dobroselsky (Dobroselsky, 2017/2021) mainly used for handling MIDI files for composition or recording with MIDI devices. One example of such is most modern keyboards. In Figure 27, we see the characteristic logo present in the documentation of the library.

In Figure 28, we see an example of code; this code will create MIDI files with notes with the desired characteristics (random duration/position). It is obvious why this library is helpful for this project, with just a few lines of this code it will create over 6500 MIDI files, in fact in less than 15 seconds. The actual code is more complex than in this figure. One particular aspect that stands out in Figure 28 is that the start had to be 5 seconds late for a reason that we will see in the TIMIDITY library.

Most of the potential that this library has is not being used because it has a conversion from actual tempo to seconds, and for this purpose the tempo is not used. This is the main complexity with the MIDI standard.



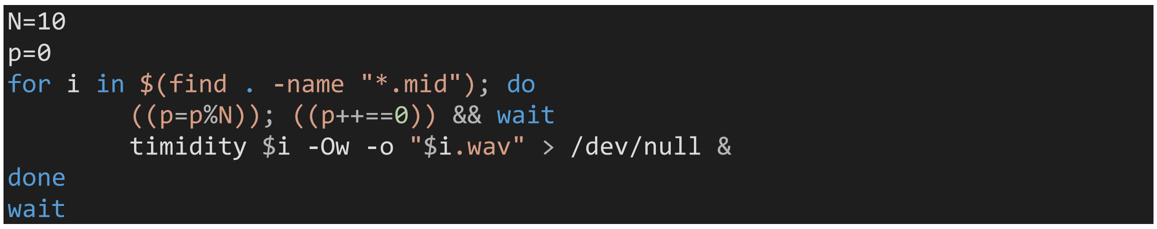
```
int[] basicnotes = { 60, 62, 64, 65, 67, 69, 71 };
string[] stringnotes = { "C", "D", "E", "F", "G", "A", "B" };
List<string> remember = new List<string>();
Random r = new Random();
for (int i=0; i!=7; i++){
    Console.WriteLine(i);
    Directory.CreateDirectory("Path//" + stringnotes[i] + "//");
    for(int j=0; j!=929; j++){
        var midiFile = new MidiFile();
        TempoMap tempoMap = midiFile.GetTempoMap();
        var trackChunk = new TrackChunk();
        using (var notesManager = trackChunk.ManageNotes()){
            NotesCollection notes = notesManager.Notes;
                    0,
                    tempoMap)));
            notes.Add(new Note(
            (SevenBitNumber)basicnotes[i],
                LengthConverter.ConvertFrom(
                    new MetricTimeSpan(hours: 0, minutes: 0,
                    seconds: 0, milliseconds: r.Next(50,1500)),//Duration
                    0,
                    tempoMap),
                LengthConverter.ConvertFrom(
                    new MetricTimeSpan(hours: 0, minutes: 0,
                    seconds: 0, milliseconds: r.Next(5500,10000)),//Moment of start
                    0,
                    tempoMap)));
        }
        midiFile.Chunks.Add(trackChunk);
        midiFile.Write("Path//" + stringnotes[i] + "//" + RandomNonrepeatingString(remember)+".mid");
    }
}
```

Figure 28. Example of code for the generation of MIDIs

4.2.2 TIMIDITY

After creating the midi files, I input them into the TIMIDITY library (Izumo, 2018). This library is a software synthesizer to play MIDI files. Synthesizers generate wave sounds depending on the instrument, the most everyday things where we see synthesizers are electronic keyboards, which, most of the time, could play different instruments. Software synthesizers create sound, but without specialized hardware, most modern software synthesizers are based on classical physical synthesizers like the Roland MT-32.

The reason for choosing this library is that very few libraries convert files from MIDI to audio without using a graphical interface. Therefore it could be done for all files in a simple script. The only feasible options were Fluidsynth and TIMIDITY. Fluidsynth should be a better option but using the default options, but TIMIDITY had better sound. In theory, using Fluidsynth, you could get better sound, TIMIDITY was already realistic, and it was highly difficult to use Fluidsynth with personalized options.



```
N=10
p=0
for i in $(find . -name "*.mid"); do
        ((p=p%N)); ((p++==0)) && wait
        timidity $i -Ow -o "$i.wav" > /dev/null &
done
wait
```

Figure 29. Code used for converting MIDI files to .wav files with TIMIDITY

I used the TIMIDITY library in Linux because creating a Shell script for using the library for all files was easier for me, but it could be used in Windows. Figure 29 is the code for converting all the MIDI files in the sub-folders to .wav files. For faster times, this code will parallelize in the CPU. In this case, I have established ten simultaneous processes converting MIDI files because it ended with better results. In fact, the process is setting batches of processes to run at the same time and then waits for all of them.

In the end, this script converted 6500 MIDI files in less than 15 minutes. This activity is more time-consuming than creating the MIDI files, but these files need to have more information, its size is more than 20000 times greater than the MIDI files.

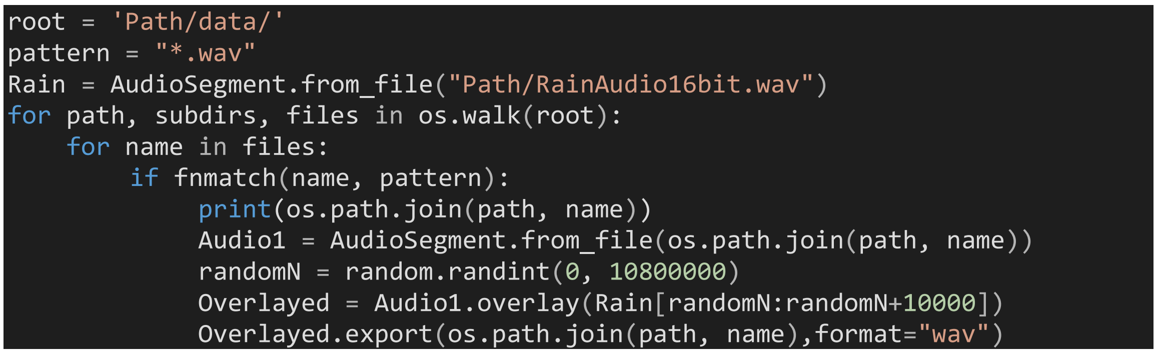
The main problem that occurred with this library was that the time before the first note was eliminated, and in the first implementation all notes started at the same time. As this problem could not be resolved, I decided to put a short "phantom note" (50ms) initially and delay the start of the actual note. Therefore, the note started at least after 5 seconds, after that there was no residual information about this initial note that could confuse the ANN.

4.2.3 Preprocessing for Background Noise

In half of the tests, the notes were just alone without any other sound. So, I have introduced random rain audio without loops for making audio that does not depend on just the generated audio and has some background noise.

I found Nik (Nik Rijavec, 2019) that had recorded 3 hours of audio of rain in the forest without loops. For me, this was impressive because finding long audios of background noise without loops was quite hard. I have found that in many places you can generate background audio noise, but this could not be reliable because perhaps has patterns that the ANN could understand easily.

I used the library Pydub (Robert, 2011/2021) for editing the audios. I chose this library because it had easy use for editing audio files. I have added not much code from this library, but I used Pydub in slicing the first "phantom note" and in a lot of testing with changing the volume of the background noise. The code present in Figure 30 just takes all the files that comply with the pattern (.wav) in the specified directory for converting them. The exciting part is that it takes a random piece of the rain audio, it would serve no purpose just to take always the same segment (the ANN would learn to ignore it as it would have seen it many times). This code is executed in less than 10 minutes.



```
root = 'Path/data/'
pattern = "*.wav"
Rain = AudioSegment.from_file("Path/RainAudio16bit.wav")
for path, subdirs, files in os.walk(root):
    for name in files:
        if fnmatch(name, pattern):
            print(os.path.join(path, name))
            Audio1 = AudioSegment.from_file(os.path.join(path, name))
            randomN = random.randint(0, 10800000)
            Overlayed = Audio1.overlay(Rain[randomN:randomN+10000])
            Overlayed.export(os.path.join(path, name),format="wav")
```

Figure 30. Code for mixing the .wav files with the random rain sound.

4.3 TensorFlow

TensorFlow (Figure 31) is one of the most advanced and versatile general-purpose libraries for developing ANNs. It was created by the Google Brain Team (Abadi et al., 2016) and was released as a free, open-source software library, and this was one of the critical successes that this library has nowadays. Its use is primarily exclusive to Python but supports mobile execution, JavaScript, and fast execution in C++ and CUDA.

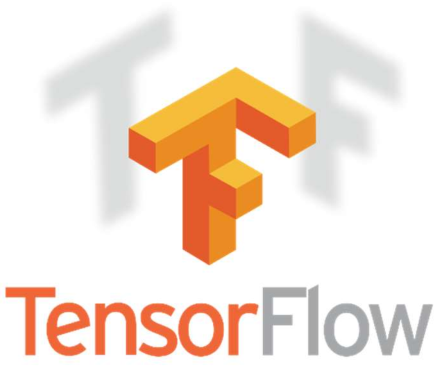


Figure 31. Logo of TensorFlow (Begoon, 2018)

I decided to use this library weighing many factors: It is a versatile library compared with other libraries in the field, with excellent performance (as I have a CUDA compatible GPU). TensorFlow could implement the complex functions of training with little effort, with the possibility of error-processing. The only drawback of TensorFlow is that it is made for easy development, which means that it is used in Python, which could lead to less performance. But in benchmarks, TensorFlow does not suffer a lot from this fact, indeed just complex libraries can outperform TensorFlow. The most appealing factor of TensorFlow is the amount of community it possesses, which helps considerably in learning this vast library.

I used this library to develop the CNN and its training: Structure of the CNN, Activation Functions, Training with SGD, and Testing with Unseen Data.

4.3.1 The Structure of the ANN

The definition of the whole structure of the final CNN is seen in Table 2, the layers are ordered from the top/input layer to the bottom/output layer. I have also decided to put the Links, but many values will be copied/moved/compared, but only the Links can be trainable (weights and bias). I will explain how this structure works, dividing the structure into two parts: Input with Convolutional and Flat with Output. I came up with this structure basing on similar examples (TensorFlow, n.d.), but in general, for taking 3D information (like an image) for classification, in most cases, the structure mainly changes in the number of layers.

When defining the Network structure, I used the ReLU activation as the activation function for all the Links. I made this decision as the ReLU is one of the most used functions when training CNNs and the easy computation (contrary to its variations) made it appealing for the number of tests that will be performed.

Layer	Size (Width x Height x Depth)	Nº of Links
Input Layer/Spectrogram	1204 x 257 x 1	0
Resizing	128 x 128 x 1	0
Convolutional 2D Nº Filters=32, Kernel Size=3x3 Stride=1x1, No Zero Padding	126 x 126 x 32	320
Convolutional 2D Nº Filters=64, Kernel Size=3x3 Stride=1x1, No Zero Padding	124x124x64	18496
MaxPooling 2D Pool size=2x2, Stride=2x2	62x62x64	0
Dropout 25%	62x62x64	0
Flatten	246016x1x1	0
Dense	128x1x1	31490176
Dropout 50%	128x1x1	0
Output	7 or 63	8127

Table 2. Complete definition of the CNN

I generated diagrams for visualizing the layers (LeNail, 2019), but the only layer that is not present in those is the Dropout layer (Hinton et al., 2012). This layer was invented just to address the problem that occurs when the ANN memorizes its training data without Learning (also called Overfitting), therefore it is not helpful in unseen data. This layer deactivates a proportion of neurons on the previous layer during training, in this case, 25% and 50%. It intends to make the ANN not dependent on some localized feature and force it into learning a more overall pattern. In this case, Dropout has been included in Table 2 because TensorFlow internally creates another layer of the same size. Nowadays, this technique is valuable, and it is used in a significant amount of ANN, but Google has the patent (United States Patent No. US9406017B2, 2016).

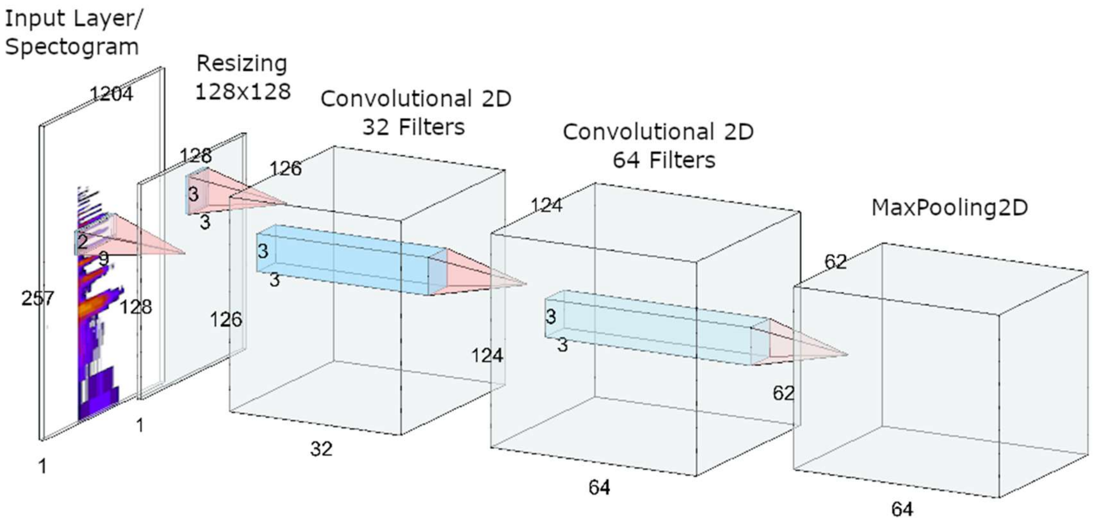


Figure 32. Input and Convolutional structure of the ANN, the Convolutional part of the ANN

The first part of the CNN that we must examine is in Figure 32. In this structure, we see all the convolutional layers. Firstly, we see the input audio (spectrogram) being resized into a shape of 128x128, this resize is commonly done in CNNs to have a simpler Network with fewer nodes, most of the time the ANN doesn't need all the information (raw data) so a compressed/resized version is used. Secondly, we have two Convolutional layers that will extract the essential features of the Audio, the first one with 32 filters and the second one with 64 filters. This leads to a MaxPooling2D layer, the pooling layers extract the higher values of the last layer, in this case, the filter is set to 2x2, so it resizes the previous layer to half but it maintains the same amount of filters as it is applied once per filter. At this stage, most of the data that has to be ignored (like rain) has been eliminated but the characteristic data of each category (note) is maintained.

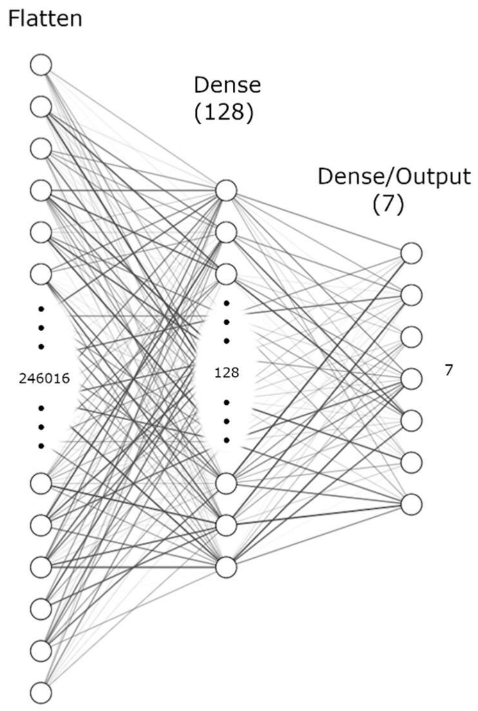


Figure 33. Structure of the ANN after the Convolutional layers with the Output

The Convolutional part of the ANN is followed by a flattened part seen in Figure 33, this part leads to the conclusion. It starts with a Flatten layer that takes all the nodes from the last layer and places them in a straight line, that's why if we take the size of the MaxPooling2D layer, we can know the size of the flatten layer (62x62x64 = 246016). This is done by interconnecting them densely to a 128 node layer, leading to the final output of the CNN also interconnected. The flattened part of the Network is the most time-consuming because most links belong to this structure, as seen in Table 2.

It could be thought that the 128 nodes Dense layer could be unnecessary, but the convolutional layers have extracted the most critical information, the Network needs to draw a conclusion from that extracted features, this means that in most cases at least one Dense layer is put in between the Flatten layer and the output. The size of this Dense could be lowered but after tests, this size seems to be appropriate.

In the Appendix, there is a complete diagram of the Network structure combining the last two diagrams.

4.4 Results with Different Datasets

For getting the results, I used the stored trained Network in TensorFlow for predicting audios that were created just for this purpose. The CNN has not used them for training or improving its capacities. To get the Graphs shown below I used a widely used library in Python called matplotlib (Barrett et. al., 2005). The number of files used for each test result was decided to be 100 test files per classifying category, this meant that the number of test files was either 700 or 6300.

For illustrating the results, we will see a Confusion Matrix, a matrix that compares the actual value with the predicted by the Network in the two axes. It is worth mentioning that in the graphs about accuracy and Loss, there are two unexplained concepts:

- Epoch: unit of measure of all the Dataset for training, it usually means the number of times that the ANN has tried to predict all available data.
- Validation Data: part of data taken from the training data that the ANN will not use for improvements, just for testing if the ANN works with unseen data. It is predicted once per epoch.

4.4.1 One Note Alone

In this case, the CNN trained with files just containing one note, with the rest of the audio silent. This is a simple problem because there is no information to remove, but this was the reason for this test, since it was the first test to prove that the CNN could learn and identify notes correctly. It may seem a simple case but as it was my first time with this technology was a good beginning for scaling and testing the Network structure.

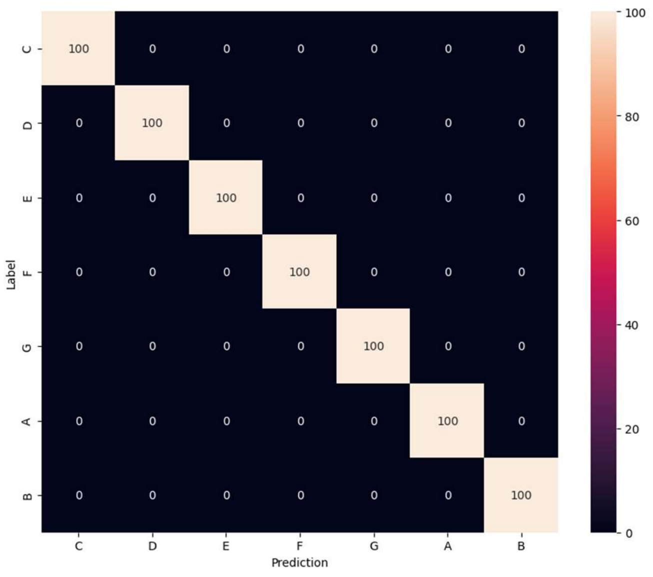


Figure 34. Confusion Matrix of the Tests with 1 Note without Background Noise

In Figure 34, we see that all the values were predicted correctly in this test, ending with an accuracy of 100%, proven with 700 generated test data. In the previous test, without changing the Network's structure to this better version, it came out about a 96/98% of accuracy.

Six thousand five hundred files for training seem to be too many, probably with 2000 files or less the CNN could get the same results. This can be analyzed from Figure 35 and Figure 36, in which we see that with just one epoch, the CNN guesses more than 95% of the validation data correctly.

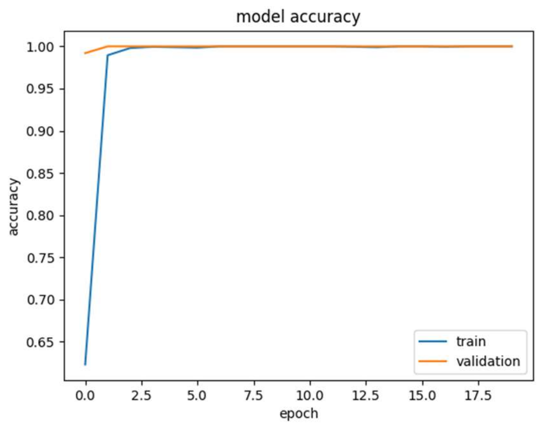


Figure 35. Accuracy of the Model - One Note Without Background Noise

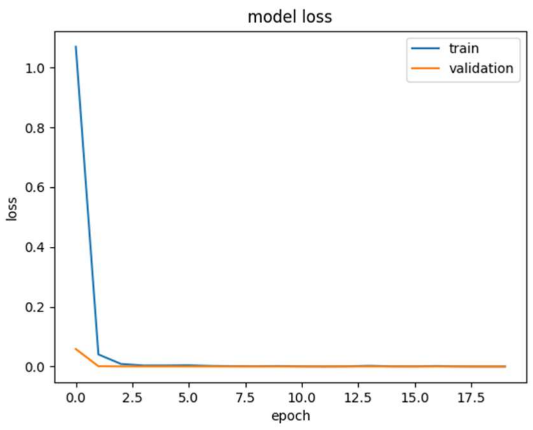


Figure 36. Loss of the Model - One Note Without Background Noise

4.4.2 One Note with Random Rain

I created this case for replicating the behaviour of the CNN in the real world, where conditions are not always perfect. As in this audio the rain sound is quite loud and can be heard over the note playing, the CNN will behave poorly or at least not guess with the same accuracy. This was not the case, as the amount of train audios per case is considerable, the CNN had a final accuracy of 100%, proven with 700 generated test data.

The Confusion Matrix was removed since it does not bring more information, it is the same as the last case. The only observable changes are the rate of accuracy and Loss, seen in Figure 37 and Figure 38, in which we see similar but slower Learning, but the amount of data builds up to almost 100% of accuracy on validation data by the 5th epoch.

To experience how the sound is generated, you can hear it on YouTube (Girbés Mínguez, 2021a).

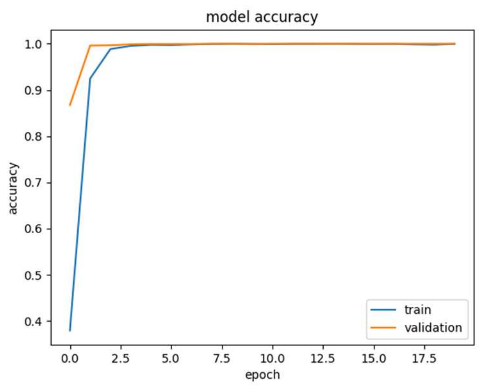


Figure 37. Accuracy of the Model - One Note With Background Noise

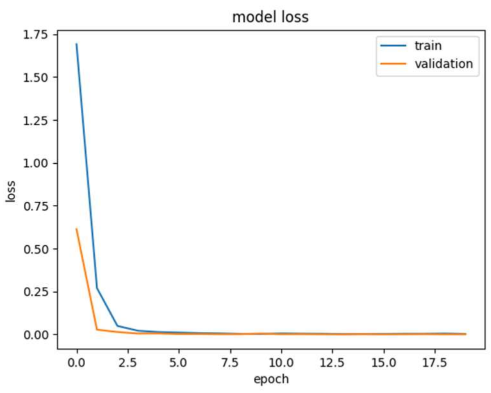


Figure 38. Loss of the Model - One Note With Background Noise

4.4.3 One, Two, or Three simultaneous Notes Alone

To see how this problem could be scaled to more piano notes or even to all piano notes, I created this case in which the individual notes are identified, and chords of two and three notes. The number of categories increased from 7 to 63, which lead to fewer train files per category (as the total amount is 6500 train files as before).

In this case, the accuracy reached 99.79%, tested with 6300 generated test files, proving that the CNN could identify more complex patterns keeping high accuracy with a more considerable number of categories. In Figure 39, we see the Confusion Matrix, it is hard to see the values that have less accuracy. Most of the times, it confused similar patterns with most of the same notes present. In nearly all cases, it thought that E-F was D-E-F or E-F-B was E-A-B.

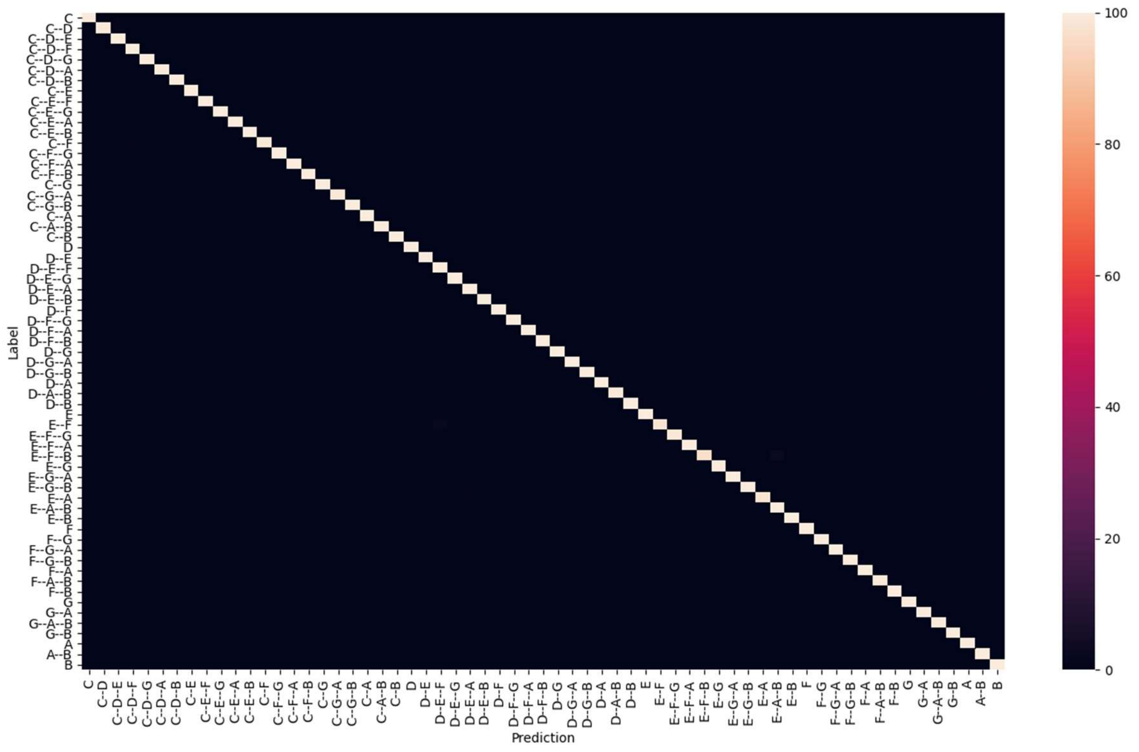


Figure 39. Confusion Matrix of the Tests with 1, 2, or 3 Notes without Background Noise

The learning, in this case, was more progressive, in contrast to the cases of one note. The lower slope in learning is due to the low quantity of training files per category, but it reached more than 90% accuracy in validation files even with just three epochs. One curious aspect of these graphs is that the validation data (unseen) behaves better than with the train data (seen). This may seem misleading, and in most cases an ANN will work the other way around, but for ANNs with Dropout it is expected. The Dropout only applies in training files, not in validation files, with them the Network uses all its potential.

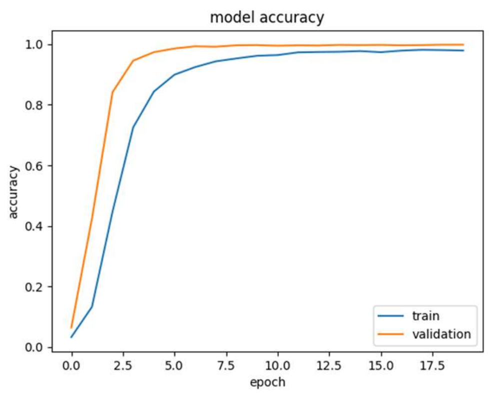


Figure 40. Accuracy of the Model - One, Two, or Three Notes Without Background Noise

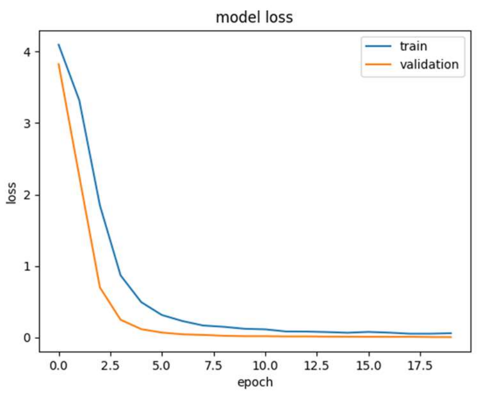


Figure 41. Loss of the Model - One, Two, or Three Notes Without Background Noise

4.4.4 One, Two, or Three simultaneous Notes with Random Rain

This became the final case and the most complex one. The CNN had the same Dataset as the last task but mixed with the rain noise. It reached a final accuracy of 96.94% in test files, as seen in Figure 42. The results in this case were less accurate, and some of the errors made by the CNN can be distinguished in the Confusion Matrix, but most of the time it confused similar cases, e. g., D-A-B instead of D-B.

In this case, the number of training files had to be doubled to 13000 audios because without that amount the CNN could not reach that accuracy. With the same data as the previous tests it resulted in around 60% of accuracy, in later stages overfitting the data. The increased Dataset for training implied that I could not get the computation brought by the GPU (NVIDIAs CUDA). So, for this case, it only used the CPU at full capacity, in the other cases, all the training occurred in less than two minutes, but here it needed more than one hour, which made the testing of the Network difficult.

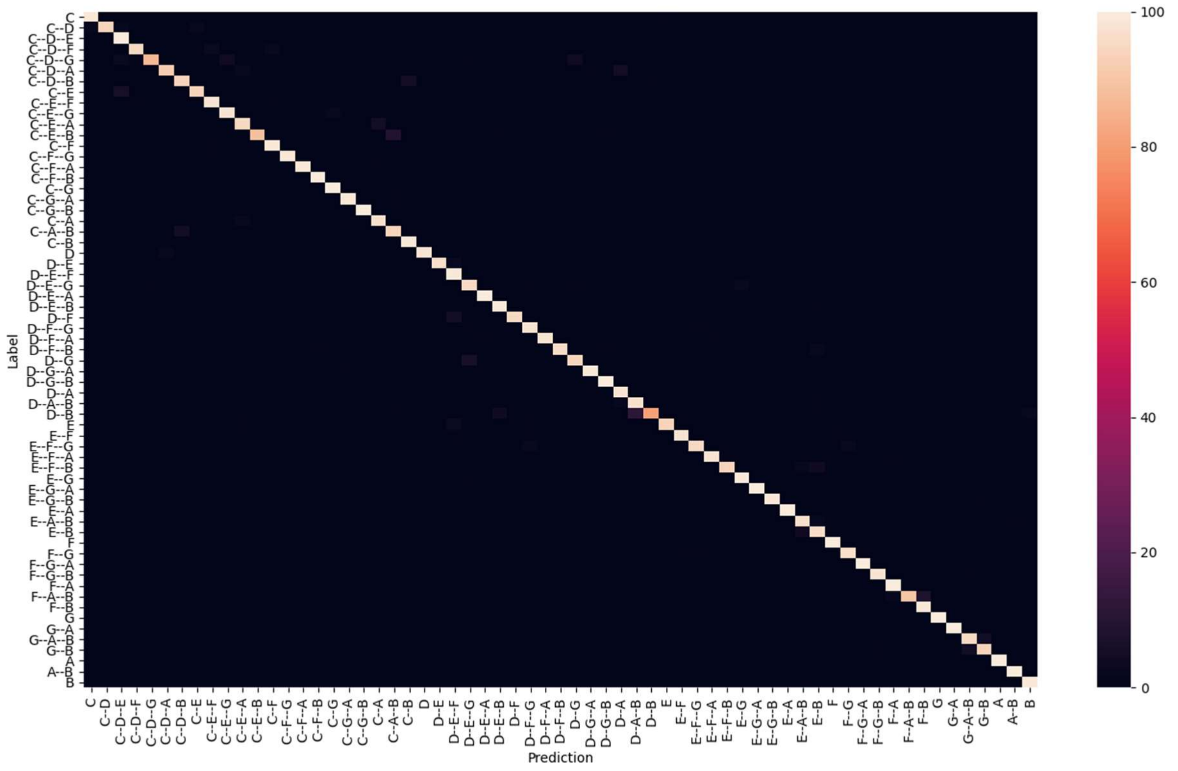


Figure 42. Confusion Matrix of the Tests with 1, 2, or 3 Notes with Background Noise

In order to appreciate how the mix of three notes with rain sound occurred, I uploaded one generated audio to YouTube (Girbés Mínguez, 2021b).

Regarding the Accuracy and Loss seen in Figure 43 and Figure 44, it is evident that it behaves much poorly in the Training files than in the previous cases. Having a more significant gap between the two training file types means that the Dropout layers are cutting down most of the Network's ability. However, with the Validation files it learned rapidly and accurately, and with five epochs it reached about 90% of accuracy.

Before the Dataset was doubled, the Overfitting of the training data could be seen in those graphs. At some point the Training and Validation curves flipped sides, with the Validation files not getting better in accuracy/loss but the training files getting more accuracy every epoch.

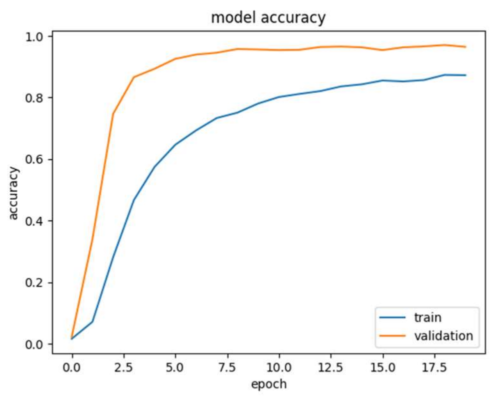


Figure 43. Accuracy of the Model - One, Two, or Three Notes With Background Noise

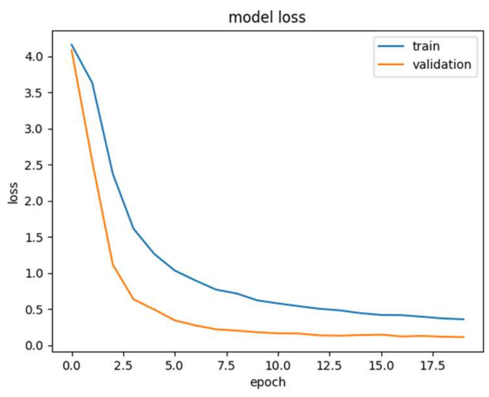


Figure 44. Loss of the Model - One, Two, or Three Notes Without Background Noise

5 Conclusion and Further Development

This project has helped me to have a practical perspective in the field of ANNs. The way that I had, on my own, to discover and work with different technologies and libraries has brought me a perspective of investigation and development typical in engineering, especially the generation of the Dataset. That said, the objective was accomplished in all means, ending with a CNN that can identify piano chords with rain noise as a background, with an outstanding accuracy with unseen data.

Regarding future extensions of this work, I think that the main step is to enlarge the Dataset. For this type of Networks to work in every possible life scenario, there may be ways of generating notes with different pianos (e.g., Grand vs Upright piano) and with an enormous amount of varying background noises (especially with people). Apart from this, I am already planning on creating a Network based on the LSTM architecture especially introducing the possibilities of audios with different lengths.

In a nutshell, I am proud of this work, especially the case results, and I look forward to working more in this field.

6 Bibliography

3Blue1Brown. (2017, November 3). What is backpropagation really doing? | Deep learning, chapter 3. Retrieved 19 May 2021, from https://www.youtube.com/watch?v=Ilg3gGewQ5U

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., & et al. (2016). Tensorflow: A system for large-scale machine learning. 12th USENIX Symposium on Operating Systems Design and Implementation OSDI 16, 265–283.

Agarwal, D. (2020, June 4). Neural Network Node. Retrieved 8 April 2021, from https://miro.medium.com/max/640/1*sPg-0hha7o3iNPjY4n-vow.jpeg

Agatonovic-Kustrin, S., & Beresford, R. (2000). Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. Journal of Pharmaceutical and Biomedical Analysis, 22(5), 717–727.

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. 2017 International Conference on Engineering and Technology (ICET), 1–6. IEEE.

Azizan, N., & Hassibi, B. (2018). Stochastic Gradient/Mirror Descent: Minimax Optimality and Implicit Regularization | Navid Azizan-Ruhi. Retrieved 19 May 2021, from http://www.its.caltech.edu/~nazizanr/papers/SMD.html

Azodi, C. B., Tang, J., & Shiu, S.-H. (2020). Opening the Black Box: Interpretable Machine Learning for Geneticists. Trends in Genetics, 36(6), 442–455. https://doi.org/10.1016/j.tig.2020.03.005

Barrett, P., Hunter, J., Miller, J. T., Hsu, J.-C., & Greenfield, P. (2005). Matplotlib–A Portable Python Plotting Package. Astronomical Data Analysis Software and Systems XIV, 347, 91.

BasicKnoledge101. (n.d.). Outline of thought. Retrieved 19 May 2021, from https://www.basicknowledge101.com/pdf/literacy/thoughtOutline.pdf

Begoon. (2018, January 5). TensorFlow logo. Retrieved 21 May 2021, from https://commons.wikimedia.org/wiki/File:TensorFlowLogo.svg

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 157–166.

Biswal, A. (2021, February 18). Recurrent Neural Network (RNN) Tutorial for Beginners. Retrieved 14 May 2021, from Simplilearn.com website: https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

Box, E. (2019, October 24). Autoencoder in biology—Review and perspectives. Retrieved 9 April 2021, from https://encodebox.medium.com/auto-encoder-in-biology-9264da118b83

Bre, F. (2017, November). Fig. 1. Artificial neural network architecture (ANN i-h 1-h 2-h n-o). Retrieved 8 April 2021, from https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051

Brilliant, M. & S. W. (n.d.). Backpropagation | Brilliant Math & Science Wiki. Retrieved 19 May 2021, from https://brilliant.org/wiki/backpropagation/#

Brownlee, J. (2021, January 17). How to Choose an Activation Function for Deep Learning. Retrieved 30 April 2021, from Machine Learning Mastery website: https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

Carden, J., & Cline, T. (2019). Absolute pitch: Myths, evidence and relevance to music education and performance. Psychology of Music, 47(6), 890–901. https://doi.org/10.1177/0305735619856098

Cavaioni, M. (2018, June 25). DeepLearning series: Convolutional Neural Networks. Retrieved 13 May 2021, from Medium website: https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524

Chen, B. (2021, January 4). 7 popular activation functions you should know in Deep Learning and how to use them with Keras and…. Retrieved 30 April 2021, from Medium website: https://towardsdatascience.com/7-popular-activation-functions-you-should-know-in-deep-learning-and-how-to-use-them-with-keras-and-27b4d838dfe6

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). ArXiv Preprint ArXiv:1511.07289.

Delua, J. (2021, March 12). Supervised vs. Unsupervised Learning: What's the Difference? Retrieved 19 May 2021, from https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning

Dobroselsky, M. (2021). Melanchall/drywetmidi [C#]. Retrieved from https://github.com/melanchall/drywetmidi (Original work published 2017)

El Naqa, I., Li, R., & Murphy, M. J. (2015). Machine learning in radiation oncology: Theory and applications. Retrieved from https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=1016285

Elman, J. L. (1990). Finding structure in time. Cognitive Science, 14(2), 179–211.

Gartzman, D. (2020, May 9). Getting to Know the Mel Spectrogram. Retrieved 14 May 2021, from Medium website: https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0

Gharakhanian, A. (2016, December 19). Generative Adversarial Network. Retrieved 27 April 2021, from https://www.linkedin.com/pulse/gans-one-hottest-topics-machine-learning-al-gharakhanian?trk=pulse_spock-articles

Girbés Mínguez, J. (2021a, May 29). Audio sample one note with background noise for thesis. Retrieved 29 May 2021, from https://www.youtube.com/watch?v=pJ-X6OmO470&ab_channel=juangirbes

Girbés Mínguez, J. (2021b, June 1). Audio sample one, two and three notes with background noise for thesis. Retrieved 1 June 2021, from https://www.youtube.com/watch?v=7LiV1EetaWc&ab_channel=juangirbes

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., … Bengio, Y. (2014). Generative adversarial networks. ArXiv Preprint ArXiv:1406.2661.

Hinton, G. E., Krizhevsky, A., Sutskever, I., & Srivastva, N. (2016). United States Patent No. US9406017B2. Retrieved from https://patents.google.com/patent/US9406017B2/en

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. ArXiv:1207.0580 [Cs]. Retrieved from http://arxiv.org/abs/1207.0580

Hui, J. (2020, July 3). Machine Learning—Hidden Markov Model (HMM). Retrieved 20 May 2021, from Medium website: https://jonathan-hui.medium.com/machine-learning-hidden-markov-model-hmm-31660d217a61

Izumo, M. (2018, August 29). TiMidity++. Retrieved 14 May 2021, from SourceForge website: https://sourceforge.net/projects/timidity/

Jordan, J. (2017, July 4). Neural networks: Activation functions. Retrieved 27 April 2021, from https://www.jeremyjordan.me/neural-networks-activation-functions/

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale video classification with convolutional neural networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1725–1732.

Karras, T., Laine, S., & Aila, T. (2019). A Style-Based Generator Architecture for Generative Adversarial Networks. ArXiv:1812.04948 [Cs, Stat]. Retrieved from http://arxiv.org/abs/1812.04948

Keim, R. (2019, November 24). Simple Perceptron. Retrieved 27 April 2021, from https://www.allaboutcircuits.com/uploads/articles/how-to-train-a-basic-perceptron-neural-network_rk_aac_image1.jpg

Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-Normalizing Neural Networks. ArXiv:1706.02515 [Cs, Stat]. Retrieved from http://arxiv.org/abs/1706.02515

Kohl, N. (2010, March 23). machine learning—What is the role of the bias in neural networks? Retrieved 28 April 2021, from Stack Overflow website: https://stackoverflow.com/questions/2480650/what-is-the-role-of-the-bias-in-neural-networks

Korf, R. E. (1996). Artificial Intelligence Search Algorithms. In Algorithms and Theory of Computation Handbook. CRC Press.

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.9220&rep=rep1&type=pdf

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient backprop. In Neural networks: Tricks of the trade (pp. 9–48). Springer.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324.

Leijnen, S., & Veen, F. van. (2020). The neural network zoo. Multidisciplinary Digital Publishing Institute Proceedings, 47(1), 9.

LeNail, A. (2019). NN-SVG: Publication-Ready Neural Network Architecture Schematics. Journal of Open Source Software, 4, 747. https://doi.org/10.21105/joss.00747

Lowndes, A. B. (2015). Deep Learning with GPU Technology for Image & Feature Recognition. DOI.

Lu, H., Zhang, H., & Nayak, A. (2020). A Deep Neural Network for Audio Classification with a Classifier Attention Mechanism. ArXiv:2006.09815 [Cs, Eess, Stat]. Retrieved from http://arxiv.org/abs/2006.09815

Nik Rijavec. (2019, May 5). Wandering Around The Forest on a Rainy Day / Relaxing Rain Sounds (3 Hours No Loop). Retrieved 14 May 2021, from https://www.youtube.com/watch?v=tfO0lH3erCQ

Noriega, L. (2005). Multilayer perceptron tutorial. School of Computing. Staffordshire University.

NVIDIA. (2018, April 23). Convolutional Neural Network (CNN). Retrieved 28 April 2021, from NVIDIA Developer website: https://developer.nvidia.com/discover/convolutional-neural-network

NVIDIA Blog. (2018, August 2). NVIDIA Blog: Supervised Vs. Unsupervised Learning. Retrieved 18 May 2021, from The Official NVIDIA Blog website: https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/

Olah, C. (2015, August 27). Understanding LSTM Networks—Colah's blog. Retrieved 15 May 2021, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Ongsulee, P. (2017). Artificial intelligence, machine learning and deep learning. 2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE), 1–6. IEEE.

Premtibadiya. (2020, July 16). Music Genre Classification using RNN-LSTM. Retrieved 16 May 2021, from Medium website: https://medium.com/@premtibadiya/music-genre-classification-using-rnn-lstm-1c212ba21e06

Robert, J. (2021, May 21). Jiaaro/pydub. Retrieved 22 May 2021, from https://github.com/jiaaro/pydub (Original work published 2 May 2011)

Ruiz Ruiz, P. (2018, March 4). Intro Clustering. Retrieved 19 May 2021, from http://www.pabloruizruiz10.com/resources/Curso-Machine-Learning-Esp/5---Aprendizaje-No-Supervisado/Intro-Clustering.html

Samek, W., Wiegand, T., & Müller, K.-R. (2017). Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models. ArXiv:1708.08296 [Cs, Stat]. Retrieved from http://arxiv.org/abs/1708.08296

Satyam Raj. (2020). Image based Bird Species Identification using Convolutional Neural Network. International Journal of Engineering Research And, V9(06), IJERTV9IS060279. https://doi.org/10.17577/IJERTV9IS060279

Sharma, A. (2017, March 30). Understanding Activation Functions in Neural Networks. Retrieved 30 April 2021, from Medium website: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

Sharma, Sagar. (2017). Activation functions in neural networks. Towards Data Science, 6.

Sharma, Shikhar. (2019, December 14). A Visual Introduction to Neural Networks. Retrieved 27 April 2021, from Medium website: https://towardsdatascience.com/a-visual-introduction-to-neural-networks-68586b0b733b

Sharma, Siddharth, Sharma, S., Scholar, U., & Athaiya, A. (2020). ACTIVATION FUNCTIONS IN NEURAL NETWORKS. 4(12), 7.

Stanford University, C. (2021). CS231n Convolutional Neural Networks for Visual Recognition. Retrieved 30 April 2021, from https://cs231n.github.io/neural-networks-1/

TensorFlow. (n.d.). Image classification | TensorFlow Core. Retrieved 22 May 2021, from https://www.tensorflow.org/tutorials/images/classification

V Srinivasan, A. (2019, September 7). Stochastic Gradient Descent—Clearly Explained !! | by Aishwarya V Srinivasan | Towards Data Science. Retrieved 19 May 2021, from https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31

Versloot, C. (2019, October 15). Leaky ReLU: Improving traditional ReLU. Retrieved 1 May 2021, from MachineCurve website: https://www.machinecurve.com/index.php/2019/10/15/leaky-relu-improving-traditional-relu/

Wood, T. (2019, May 17). What is the Softmax Function? Retrieved 5 May 2021, from DeepAI website: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. ArXiv Preprint ArXiv:1505.00853.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. European Conference on Computer Vision, 818–833. Springer.

Zilouchian, A. (2001). Fundamentals of neural networks. Intelligent Control Systems Using Soft Computing Methodologies, (1), 1–5.

Index of Figures

Index of tables

Appendix

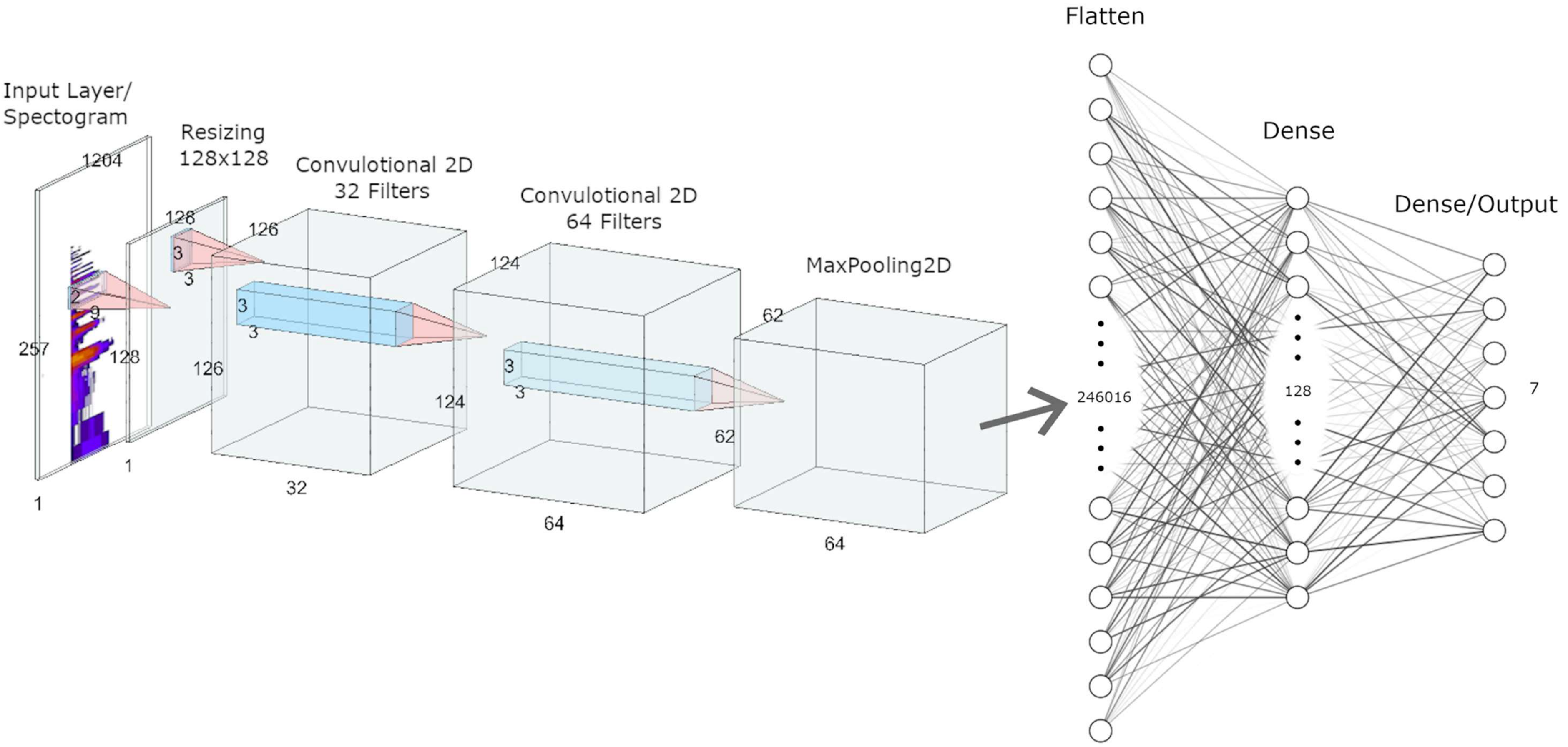


Diagram of the complete structure of the CNN used for Piano Note Recognition