BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS

Machine Learning in Sound Classification

Supervisor: Lect. Dr. Radu-Lucian LUPŞA

Author: Abrudan Andrei

2019

# Contents

[Contents 2](#_Toc18197087)

[References 3](#_Toc18197088)

[Abstract 5](#_Toc18197089)

[Introduction 5](#_Toc18197090)

[Background 6](#_Toc18197091)

[Neural Networks 6](#_Toc18197092)

[Neurons 6](#_Toc18197093)

[Layers 7](#_Toc18197094)

[Batch size 8](#_Toc18197095)

[Epoch 8](#_Toc18197096)

[Overfitting 8](#_Toc18197097)

[Underfitting 8](#_Toc18197098)

[Accuracy 8](#_Toc18197099)

[Loss 8](#_Toc18197100)

[Stochastic Gradient Descent 9](#_Toc18197101)

[Activation function 9](#_Toc18197102)

[Fuzzy Logic 10](#_Toc18197103)

[TensorFlow 10](#_Toc18197104)

[Batch Normalization 11](#_Toc18197105)

[2D Convolution Layer 12](#_Toc18197106)

[Max Pooling 13](#_Toc18197107)

[Flatten 13](#_Toc18197108)

[Dense 14](#_Toc18197109)

[Dropout 14](#_Toc18197110)

[How the program works 14](#_Toc18197111)

[Audio to Spectrogram 14](#_Toc18197112)

[Generating the Dataset 15](#_Toc18197113)

[Designing the Model 15](#_Toc18197114)

[Compiling the Model 17](#_Toc18197115)

[Optimizer: RMSprop 17](#_Toc18197116)

[Loss: categorical\_crossentropy 17](#_Toc18197117)

[Metric: accuracy 17](#_Toc18197118)

[Deploying the Model 17](#_Toc18197119)

[Conclusion 18](#_Toc18197120)

# References

Bushaev, V. (2018, September 2). *Understanding RMSprop — faster neural network learning*. Retrieved from Towards Data Science: towardsdatascience.com/62e116fcf29a

Changhau, I. (2017, June 7). *Loss Functions in Neural Networks*. Retrieved from isaacchanghau.github.io/post/loss\_functions/

Chollet, F. (2016, January 30). *How convolutional neural networks see the world*. Retrieved from keras.io: blog.keras.io/how-convolutional-neural-networks-see-the-world.html

Dabbura, I. (2018, April 1). *Coding Neural Network — Forward Propagation and Backpropagtion*. Retrieved from Towards Data Science: towardsdatascience.com/coding-neural-network-forward-propagation-and-backpropagtion-ccf8cf369f76

Glorot, X., Bordes, A., & Bengio, Y. (2011). *Rectifier and softplus activation functions. The second one is a smooth version of the first.* AISTATS. Retrieved from proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

Hinton, G., Krizhevsky, A., Sutskever, I., & Srivastva, N. (2013). *United States Patent No. US9406017B2.* Retrieved from patents.google.com/patent/US9406017B2/en

Keras. (n.d.). Retrieved from keras.io

Popular networks. (2018, November 20). *VGG16 – Convolutional Network for Classification and Detection*. Retrieved from Neurohive: neurohive.io/en/popular-networks/vgg16/

Rouse, M. (2016, August). *Definition Fuzzy Logic*. Retrieved from searchenterpriseai.techtarget.com/definition/fuzzy-logic

Sound eXchange. (n.d.). *Sound eXchange*. Retrieved from HomePage: sox.sourceforge.net

Student created article. (2018, February 27). *Max-pooling*. Retrieved from Computer Science Wiki: https://computersciencewiki.org/index.php/Max-pooling\_/\_Pooling

TensorFlow. (n.d.). Retrieved from tensorflow.org

Unruh, A. (2017, November 10). What is the TensorFlow machine intelligence platform? Retrieved from Opensource.com: https://opensource.com/article/17/11/intro-tensorflow

Wikipedia contributors. (2019, June 11). *Dropout (neural networks)*, 901427811. Retrieved from Wikipedia: en.wikipedia.org/w/index.php?title=Dropout\_(neural\_networks)&oldid=901427811

Wikipedia contributors. (2019, August 1). *Softmax function*. Retrieved from Wikipedia: wikipedia.org/w/index.php?title=Softmax\_function&oldid=908788637

Wikipedia contributors. (2019, August 27). *Wikipedia, The Free Encyclopedia*. Retrieved from wikipedia.org/w/index.php?title=Spectrogram&oldid=912782480

# Abstract

Until recently, the task of examining and classifying sounds was done exclusively by humans. The reason is that automation could simply not cope with the amounts of data that make up an audio file. Furthermore, it is also meaningless to study sounds in vitro, so to speak, where every factor is crispy clear. For such a machine – able to classify music, speech or even background noise – to be reliable, it must be deployable in real life situations, where its receptors are often bombarded with a variety of sound waves.

Thankfully, with the introduction of neural networks (NN) and more specifically convolutional neural networks (CNN), this task is now at least approachable, if not solvable.

Working with raw audio data remains difficult, however. Because of this, there is a preliminary step involved: converting the sound to a spectrogram.

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time[[1]](#footnote-1). It allows us to abstract from the audio a simplified image that is easy to work with.

Using the image, it becomes more manageable to process and classify the sound encoded and the neural network can take advantage of built-in mechanisms

# Introduction

Machine learning has become an increasingly important field in computer science. Without the use of a neural network, a scientist would have resort to applying a variety of heuristics to a dataset in order to extract the relevant information within. Moreover, by distorting the data, relevant information may be lost resulting in false positives or other mistakes to be made. On top of this, heuristics themselves are by no means easy to invent, much less implement.

Because of this, the necessity of a system which could be fed raw data (or with as few transformations as possible) arose. Thankfully, neural networks came along and provided this exact advantage over traditional applications.

The topic of this paper will focus on how machine learning can be used to classify sounds.

The reason why this is such an interesting topic is because humans are really good at breaking down and classifying sounds (even in a conversation where there is a lot of background noise, we can distinguish words from everything else), but machines have a hard time doing the same.

The general overview of what we will achieve is:

* Extract a simplified version of the audio in what is known as a spectrogram
* Train a neural network to classify the spectrograms
* Use the classifier to label new sounds

Going forward, we will be taking a look at what a neural network is, what parameters we can change to improve both the performance and quality of the neural network, what are the building blocks of a neural network, and how the application was made with respect to all previous information.

# Background

## Neural Networks

A neural network (NN) is a set of algorithms that attempts to model the human brain in an effort to recognize patterns. These patterns are numerical and work with real numbers. Because of this, a weighting system is applied to determine the strength of a connection between two nodes resulting in fuzzy logic[[2]](#footnote-2), much like a biological brain works. Neural networks come in two major types:

1. **Convolutional Neural Networks (CNN)** which are used to classify (or label) images
2. **Recurrent Neural Networks (RNN)** used in predicting data that follows a trend

though there are a variety of other ways they can be applied, such as in:

* **Multilayer Perceptrons** used to classify data that cannot be separated linearly
* **Modular Neural Networks** which feature different networks that function independently and do not interact with each other during the computation process
* **Feedforward Neural Network** mostly used in face recognition

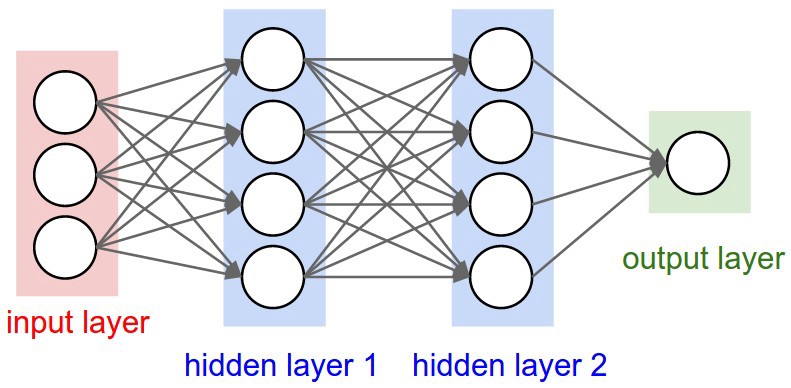
****

Figure 1 - Neural Network with two hidden layers[[3]](#footnote-3)

### Neurons

These networks are made up of neurons which take some input, apply a mathematical formula to it and produce an output. The formula that is applied is known as an *activation function*.

### Layers

The neurons are stacked over each other in groups which are called *layers*. All neural networks have at least 3 layers, which are:

#### Input layer

There is exactly one input layer. Its role is to introduce the data into the neural network for further processing. The selection of this layer depends on the shape of the data. For example, if we want to process an image, we would have a 2D convolution layer as the input layer and the input shape would be a tuple that contains the number of images in a batch, the width of an individual image, its height and the number of channels each pixel contains (i.e. 1 if the image is greyscale, 3 if it is RGB etc.)

#### Hidden layers

There can be as many hidden layers as we want. They are used to transform the data such that the neural network can make sense of it. They can be thought of as prisms through which the data changes and stimulates more and more only certain neurons that lead to a specific outcome

#### Output layer

Again, there is only one output layer. It will output a prediction based on the data it receives. The prediction is not a “black or white” outcome, but the neural network’s certainty that the result is the correct one. For well trained networks, the certainty gets closer and closer to 100%

Although we have mentioned that there can theoretically be an infinite number of hidden layers, their number can drastically affect the performance (in terms of speed) of the model, and stacking as many layers on top of each other as the machine will allow, does not necessarily result in a better model either. It is crucial that the layers follow a specific idea of how we want the data to be interpreted and as such, the choice of layers in the model becomes the most important part of constructing a reliable network. Later on, we will take a look at the predefined layers at our disposal provided by TensorFlow and what each layer helps us achieve.

The number of neurons in a specific layer is proportional to the layer’s place in the neural network. For good results and manageable consumption of resources, layers closer to the input layer should have a relatively small number of neurons. As they get closer to the output layer, this number increases. This culminates with the output layer, which generally has less neurons than the layer previous to it and can be determined by the type of neural network that is constructed:

* In a CNN, the number of neurons in the output layer is equal to the number of classes (labels) we have. For example, if we want to label a picture as either dog, cat or bird, the output layer will have exactly 3 neurons; even though the previous layer might have had as many as 1024
* In an RNN, the output layer always has exactly 1 neuron. This is because an RNN is expected to have only one output which is the data that best fits the trend.

As a rule of thumb, the number of neurons should be multiples 2.

## Batch size

This is the number of samples that will be passed through the neural network at one time. Generally, the larger the batch size is, the quicker the model will complete an epoch during training. This is because a machine may be able to process more than one sample at a time. The tradeoff, however, is that even if our machine may be able to handle very large batches, the quality of the model may degrade as we set our batches to be larger.  
This parameter should be tested and tuned based on how the model is performing during training. Not only that, the batch size will also have to be tuned in accordance with resource utilization as it affects both memory and processing power consumption.

## Epoch

An epoch refers to the number of times we train the neural network with the same input data. Of course, the data is shuffled between each epoch, otherwise it would quickly lead to overfitting. It is desired to have more than one epoch such that our datasets do not have to contain millions of records. The drawback, however, is that setting a large number of epochs will also cause overfitting. At some point, the model will stop generalizing the features of each sample, but instead generalize the sample itself leading to very good training results and very poor predictions or testing results.

## Overfitting

Overfitting occurs when our model becomes very good at classifying or predicting data that was in the training set but performs worse when working with data that it was not trained on. If the validation metrics are considerably worse than training metric, then that indicates that our model is overfitting. Similarly, we can tell that the model is overfitting if during training the metrics were good, but when we use the model to predict on test data it is not accurately classifying the data in the test set. The concept of overfitting boils down to the fact that the model is unable to generalize well

## Underfitting

A model is said to be underfitting when it is not even able to classify the data it was trained on well; let alone data it has not seen before. We can tell that the model is underfitting when the metrics given for the training data are poor.

## Accuracy

Accuracy is the main metric by which we measure the quality of our model. This value represents the certainty with which the model predicted what is the correct label for some input. Accuracy can only be applied to classification tasks.

## Loss

The loss function is a metric similar to accuracy which the Stochastic Gradient Descent (SGD) tries to minimize by continuously updating the weights and the model during training. During the training process, at the end of each epoch, the loss will be calculated on the model’s predictions. To obtain this value, the model calculates the error on each input by looking at what output it predicted for it and taking the difference between that value and its correct label.  
The following are some commonly used loss functions:

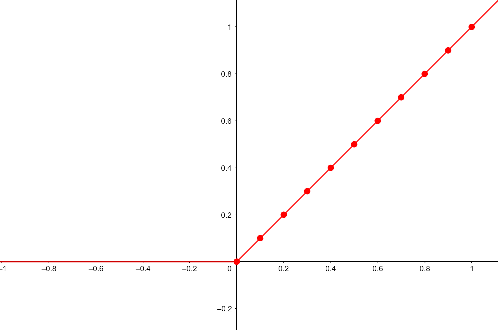
* **Mean Squared Error (MSE)** […] Widely used in linear regression as the performance measure
* **Mean Squared Logarithmic Error (MSLE)** […] used to measure the difference between actual and predicted
* **Mean Absolute Error (MAE)** is a quantity used to measure how close forecasts or predictions are to the eventual outcomes
* **Mean Absolute Percentage Error (MAPE)** is a variant of MAE
* **Kullback Leibler (KL) Divergence** […] is a measure of how one probability distribution diverges from a second expected probability distribution
* **Poisson loss function** is a measure of how the predicted distribution diverges from the expected distribution
* **Cross Entropy** is commonly-used in binary classification (labels are assumed to take values 0 or 1) (Changhau, 2017)
* **Categorical Cross Entropy** is used for single label categorization. This is when only one category is applicable for each data point
* **Sparse Categorical Cross Entropy** same as Categorical Cross Entropy, but labels are mutually exclusive

## Stochastic Gradient Descent

## Activation function

An activation function is simply a mathematical formula that takes a numerical input and outputs another.

Figure 2 - ReLU

In our model, we will be using **Rectified Linear Unit (ReLU)** activation. This is because a study has concluded that deep sparse rectifier networks are more beneficial to image classification tasks than logistic sigmoid neurons[[4]](#footnote-4). The function can be seen in Figure 2 which can be obtained by plotting

Another activation function that we will be using is **Softmax**. In mathematics, the softmax function, also known as softargmax or normalized exponential function, is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers[[5]](#footnote-5). The formula for for the softmax function is   
As the definition might suggest, we will be using softmax in the output layer to obtain a probability distribution which will represent the certainty with which the model thinks an input should have a certain lable

## Fuzzy Logic

Fuzzy logic is an approach to computing based on "degrees of truth" rather than the usual "true or false" (1 or 0) Boolean logic on which the modern computer is based[[6]](#footnote-6). The reason why this type of logic is useful is because it is so similar to the way biologic neurons work: the stimulation of a nerve is not “digital” (i.e. it is not a case of stimulated vs not stimulated) but instead is an analog process (it can take an infinite amount of states, all in the space [no stimulation, maximum stimulation]).

# TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.[[7]](#footnote-7)

Figure 3 - Comparison between TensorFlow, Keras, Theano and PyTorch interest in the past 5 years

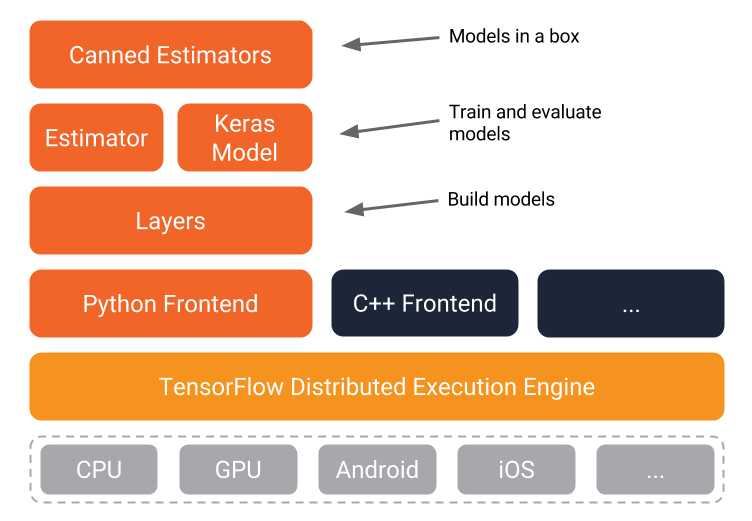
TensorFlow is one of the many machine learning APIs one might use to build a neural network. As can be seen in Figure 3, it quickly gained popularity since its launch in 2015 and still dominates the market to this day (though the graph might be biased since the data was gathered with Google Trends, and TensorFlow itself is a Google company).

Figure 4 - TensorFlow architecture

Along with TensorFlow, we will be using Keras. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.[[8]](#footnote-8)

[[9]](#footnote-9) illustrates the many ways in which we can use TensorFlow. Everything above the TensorFlow Distributed Execution Engine provides a coding style that can take advantage of the complex algorithms which manage the neural network.

As we can see, the python frontend is by far the most developed and mature with Keras being a high-level API that allows us to write succinct code.

As discussed before, we will now have a look at the different layers Keras provides us with in order to build our models:

## Batch Normalization

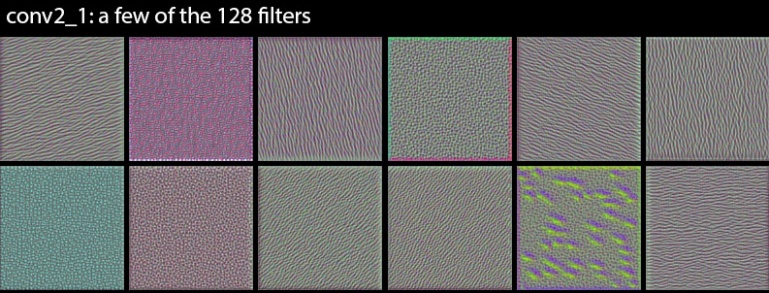
When training a neural network, we want to normalize or standardize our data in some way as part of the preprocessing step (this is a step where we get our data ready for training). The objective of normalization is to transform the data such that all the data points are on the same scale. A typical normalization process consists of scaling the numerical data down to be on a scale from zero to one.

The reason we need to normalize the data, is because we may have some numerical data points in our dataset that might be very high and others that might be very low, or we might have 2 features that are not on the same scale (for example a person’s age would vary on a scale from 1-100 vs their income which would be on a scale from $1.000-100.000). The large datapoints in an unnormalized dataset can cause instability in neural networks because the relatively large inputs can cascade down through the layers in the network and significantly decrease the training speed.

## 2D Convolution Layer

This layer allows the neural network to pick out or detect patterns and make sense of them. This pattern detection, makes these layers so useful for image analysis.

Figure 5 - 2D Conv after applying 64 filters

As the name might suggest, convolution layers are what distinguish other types of neural networks from CNNs.

With each convolution layer, we need to specify the number of filters the layer should have; these filters are what detect the patterns.

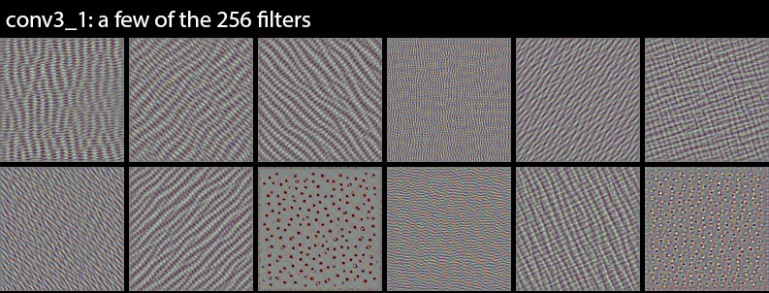
Patterns that filters can detect can be anything; some filters may be able to detect edges, others could detect corners, some may detect circles, others squares. These simple geometric shapes are what we would see at the start of the network. The deeper the network goes, the more sophisticated these filters become. So, in later layers, our filters may be able to detect specific objects like eyes, ears, hair, feathers etc. Towards the end, the filters might be able to detect even more complex objects like full dogs, cats and birds.

Figure 6 - 2D Conv after applying 256 filters

Figure 7 - 2D Conv after applying 128 filters

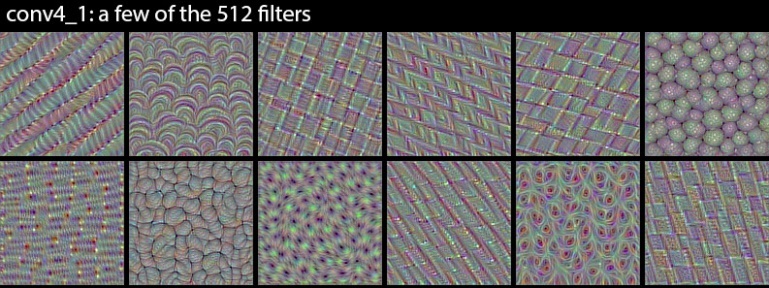
When the convolutional layer receives input, a filter of, let’s say, size 3x3 will slide over each 3x3 set of pixels from the input until it has slid over every block in the image. This sliding is referred to as convolving, so it would be more accurate to say that the filter convolves over each 3x3 block of pixels from the image. The transformation applied is the dot product between the filter and the block over which it convolves.[[10]](#footnote-10)

Figure 8 - 2D Conv after applying 512 filters

Figure 9 - 2D Conv after applying another 512 filters on top of the previous one

## Max Pooling

Max pooling is an operation that is typically added to CNNs following individual convolutional layers. When added to a model, max pooling reduces the dimensionality of images by reducing the number of pixels from the output of the previous convolution layer.

Operation wise, max pooling takes an N x N region as the filter, for example 2x2. The other parameter for this operation is the stride, meaning by how many pixels we want the filter to move as it slides across the image; for the example we will use 2 as well.

The transformation works as follows:

1. From the output of the convolution layer, we take a region that has the same size as the filter
2. We compute the maximum of that region and write it as the output
3. The region moves to the right by as many pixels as the stride specifies and repeats.  
   If we reached the end of the row, the next row begins under the current row, offset by as many pixels as the stride specifies

As we can see in Figure 10[[11]](#footnote-11), max pooling reduces the amount of data the model has to work with, thus reducing complexity and improving performance.

Figure 10 - Illustration of how MaxPooling2D works

However, because max pooling is a sort of lossy compression, it becomes dangerous to apply it to images with low resolution or in the case of very fine images where each detail is important.

There are other alternatives the operation described above (which can be referred to as Max Pooling 2D as it works with 2 dimensional objects such as images). Those are Max Pooling 1D and 3D which work as the name might suggest.

## Flatten

Flatten is part of the reshaping operations, which also include Reshape and Squeeze. These operations alter the shape of the data such that it becomes compatible with other layers we might want to use; for example, the output of a Convolution Layer cannot be used as input for a Dense layer, and so we need to reshape the data.

Flattening a tensor (i.e. the data) means to remove all of the axes, except for one. To reformulate, upon flattening any type of tensor, be it 2D, 3D or even larger, we obtain a vector that contains all the data in a sequential order.

## Dense

A dense layer represents an operation in which every input neuron is connected to every output neuron by a weight that constantly gets updated. The name is reminiscent of the densely connected graphs, which strike a resemblance with the way one might imagine the connection between a previous layer and a dense one. We have already seen an example of how dense layers look like back in Figure 1.

Dense layers can be used to reduce the number of outputs of the previous layer. Usually, the output layer is a dense one in a CNN or RNN because of the necessity to restrict the outputs to the number of classes that we have.

## Dropout

Dropout is a regularization technique patented by Google[[12]](#footnote-12) for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to dropping out units (both hidden and visible) in a neural network.[[13]](#footnote-13)

# How the program works

## Audio to Spectrogram

As stated in the introduction, our model will be used to classify sounds, based on a training dataset which provides the audio recordings along with the classes that we will be limited to.

An intermediate step appears: we must first convert the recordings into spectrograms of manageable sizes. This is because of the complexity of the audio encodings. For example, a wave file generally encodes sounds at 44.100 Hz. That means there are 44 thousand samples (which are composed of a minimum and a maximum sample) being recorded across multiple channels (as audio usually comes in stereo format) and the size of each sample depends on the encoding used when creating the file (which can be either 16 or 32 bits).

As one might imagine, this amount of data coupled with the variety of formats can easily overwhelm any neural network. For this reason, we will take advantage of Sound eXchange[[14]](#footnote-14). SoX is a command line utility that can convert various formats of computer audio files in to other formats. It also provides us with some simple options to convert the audio files to spectrograms of any length.

The length of the audio is important as a larger file would probably contain more than one class. Therefore, it is desired to separate it into smaller chunks which can be individually analyzed by the neural network.

## Generating the Dataset

Now that the data we want to process has been converted to images, we can start loading them into memory to create the dataset that will be used for training. It is important to have the data on hand, ready to be processed, as to not starve the CPU/GPU. A major improvement in this regard comes in the form of lazily loading the images, rather than all at once, though this task might prove challenging as once we pass the dataset to the training function, it is out of our hands, but doing multiple trainings will result in a poorer model.

An important step after generating the dataset is establishing the number of classes the dataset has. These classes will be the names of the items that we try to classify (for example, dog, cat or bird), but a model cannot work with strings. Instead, we have to convert the data into a numerical form, which we can achieve through the use of a LabelEncoder.

A possible improvement would be to generate additional images that have slight modifications (such as being tilted, flipped, or even cropped) in order to discourage

## Designing the Model

This part of the process will prove to be the most difficult and will require all the knowledge that has been laid down until now.

Figure 11 - The model used



Figure 11 shows the model used for this application.

We start with a Convolution Layer. This is because we are working with pictures and we are trying to find patterns that would tell us which class the corresponding sound belongs to. One parameter we have not talked about stands out: padding. Convolving the image does not only apply the filter over the image, but also has the unfortunate side effect of reducing the image size by a couple of pixels. The formula to calculate the dimensions of the output image is as follows:  
For an input and a kernel size of , the output will have a size of So, for an image of and a kernel of the resulting image will have dimensions  
This is where padding comes into play and allows us not to lose any data by extending the image (padding it) with zeroes.

One could also notice that the kernel size is quite a large number: 7. This is because we are working with relatively large images (of resolutions greater than ). At these sizes, it is recommended to have large values for the kernel for performance reasons. As we go deeper into the network, we observe that the kernel size decreases, but the number of filters we apply goes up.

The second layer normalizes the data (i.e. it translates the pixels from range to range ). This layer could have been the first just as well, but through experimentation, the results showed that the quality of the model was best in this configuration, slightly worse if normalization came after the last convolution, worse yet if it was the first layer and very poor if normalization was taken out entirely.

The next layers are two groups of Convolving and MaxPooling. Because convolution layers produce as many output images as there are filters (for example, if we apply 16 filters to a single image, we obtain 16 output images) this process is very taxing. This is why we reduce the image size by applying MaxPooling.  
In theory, we could have many more such groups. For example, VGG16, a “Very Deep Convolutional Network for Large-Scale Image Recognition”, features 13 convolution layers, but, as the article itself says: “It is painfully slow to train”[[15]](#footnote-15).

After the convolution layers follows a Dropout. As explained before, its role is to discourage overfitting. The value of the dropout depends on how well the model behaves and responds to the dropout.

The next layer is Flatten, and it takes us from the 2D territory of images, to 1D as the following layers cannot continue to work with the former shape of data.

The Dense layer featured on line 15 finally allows us to start classifying the data. I would have liked to have more than one dense layer, but unfortunately, my machine could not handle it.

The dropout featured right before the output layer has the same role as the previous one

Finally, the output layer. It is worth noting that the number of neurons in this layer is not fixed, as in previous layers, but instead is a variable which tells the model how many classes the training dataset has.  
The output of the model, will be an array of certainty which will tell us what the model thinks the result is.

## Compiling the Model



To compile the model (i.e. to get it ready for training, testing and predicting), we have to specify some parameters:

### Optimizer: RMSprop

RMSprop is an unpublished optimization algorithm designed for neural networks, first proposed by Geoff Hinton in lecture 6 of the online course “Neural Networks for Machine Learning”[[16]](#footnote-16). Without getting into too many details, RMSprop is popular enough that, although it remains unpublished, most frameworks include its implementation out of the box

### Loss: categorical\_crossentropy

As mentioned before, loss is a metric that we try to minimize. Categorical Crossentropy is a loss function that is able to work with labels that are one-hot encoded.

### Metric: accuracy

This is the metric we are interested in and as such, the output of the model will reflect this metric. We are allowed to specify other metrics, or even more than one, but needless to say, accuracy is the most important in classification problems

## Deploying the Model

In order to test and train the model, a dataset is in order. To this end, the “Free Spoken Digit Dataset” which can be found on GitHub[[17]](#footnote-17), appears to suit our needs precisely. It features recordings of people speaking the numbers 0 through 9 out loud.

Because generating the dataset also shuffles and reserve a percentage of the data for testing (through the use of function train\_test\_split, handily provided by scikit-learn, a tools library for machine learning), the next step will be simply to feed the data to the model and wait for it to be trained.

During the training process, we receive some feedback from TensorFlow; this tells us what epoch the model has reached, which batch is currently processed (and how many are left), the loss value and the accuracy of the model. Of course, the two metrics are subject to bias and will need to be compared to the testing results, but they give us an idea on how the model behaves.

After training, the testing dataset will be fed to obtain a final evaluation, from which we can deduce if the model has been well trained or if it is subject to underfitting/overfitting.

After training, we can now finally use the model to predict on new data it has never seen before.

# Conclusion

To sum up this project, I would like to point out my understanding of neural networks before and after its completion.

Before starting, I imagined neural networks as a black box which you could feed data, and it would spew out what you want out of it. This belief was cultivated through the numerous news reports I have seen throughout the years about how machine learning achieved these incredible feats, such as detecting cancer in patients with an accuracy of 97% or their deployment in autonomous cars. Perhaps these reports are not inaccurate, but they certainly hide the hard work that goes into such projects by showcasing only the results.

The reality is much different, I have discovered. The process is a tedious one; from data collection, to model building, to making sure the network has learned to detect patterns and not how to fool the reviewer. I emerge from this experience with a better understanding of what neural networks are, what are its limitations, and how we can harness their power.

The idea for the project started as a classifier for song genres. It was obviously scaled down a bit, because of difficulty and time constraints, not to mention the lack of any dataset to train the model on, but in the end, I am pleased with the result.

The model obtained is not perfect unfortunately. During testing, I have obtained good results (above 95% accuracy), but trying to classify numbers spoken out loud by a text to speech program (which do sound crispy clear in comparison with the dataset provided for training) has resulted in quite poor labels. Despite this, I am confident that by expanding the model and diversifying the training data, these problems can be overcome

1. (Wikipedia contributors, Spectrogram, 2019) [↑](#footnote-ref-1)
2. (Rouse, 2016) [↑](#footnote-ref-2)
3. (Dabbura, 2018) [↑](#footnote-ref-3)
4. (Glorot, Bordes, & Bengio, 2011) [↑](#footnote-ref-4)
5. (Wikipedia contributors, 2019) [↑](#footnote-ref-5)
6. (Rouse, 2016) [↑](#footnote-ref-6)
7. (TensorFlow, n.d.) [↑](#footnote-ref-7)
8. (Keras, n.d.) [↑](#footnote-ref-8)
9. (Unruh, 2017) [↑](#footnote-ref-9)
10. (Chollet, 2016) [↑](#footnote-ref-10)
11. (Student created article, 2018) [↑](#footnote-ref-11)
12. (United States Patent No. US9406017B2, 2013) [↑](#footnote-ref-12)
13. (Wikipedia contributors, Dropout (neural networks), 2019) [↑](#footnote-ref-13)
14. (Sound eXchange, n.d.) [↑](#footnote-ref-14)
15. (Popular networks, 2018) [↑](#footnote-ref-15)
16. (Bushaev, 2018) [↑](#footnote-ref-16)
17. <https://github.com/Jakobovski/free-spoken-digit-dataset> [↑](#footnote-ref-17)