Identifying bird species using a neural network

Computational model by:
Jonathan Anbary and Ahmir Lewin

This work amounts to two units as partial fulfillment of the requirements for receiving a grade in the subject of computational sciences.

This work was done under the instruction of Shlomo Rozenfeld.

Date:
22/06/2022

Abstract

In this work we will train a neural network to identify different species of birds based on an audio recording of the bird.
The features in this model are a spectrogram of the sound recording of the bird and the length of the recording.
We used a fully connected neural network with one hidden layer, we used the hyperbolic tangent function for the hidden layer and the sigmoid function for the output layer.
We taught the network using about thirty thousand examples of recordings of five different bird species and tested it using twenty thousand examples.
The best accuracy we achieved for distinguishing between five different bird species is 55% and when distinguishing between one species to the rest we got an accuracy as high as 90%.
important to note that when we train the network with an equal number of examples for each target we only get an accuracy of 82%.

Table of content:

Part 1 - Low resolution spectrograms

Theory

A neural network is basically a large number of connections (weights) and biases which we use to transform data (the features) to different data (the outputs).
A network is composed of several layers with the first being the input layer the last being the output layer and in between we have hidden layers. The input layer contains the features which we feed to the network, and every neuron in a subsequent layer is equal to the activation function of a weighted sum of all the previous neurons plus a bias. The reason we use an activation function is to introduce non-linearity to the network.
Eventually we will get some result in the neurons of the output layer which will tell us how the network identified the particular example we gave it. Teaching the network consists of changing the weights and biases so that the result the network gives in the output layer is closer to the result we want for any particular example (what we are actually trying to do is to minimise the cost function).
We perform these changes in the process of backpropagation.
After running all of our examples through the network we save the results and start going backwards through the network to calculate the gradients.
The gradients are calculated by deriving the cost function with weights and biases of different layers as inputs.
Because of the chain rule we don't have to recalculate the derivative and instead only have to multiply it by the derivative of the previous layer (going from output to input since the output is “closest” to the cost function) with the current layer as input.

$$\frac{dC(Z1)}{dZ1} = \frac{dC(Z2)}{dZ2} \cdot \frac{dZ2(Z1)}{dZ1}$$

We update the weights and biases based on the gradients, and the learning rate and repeat the process.
In our particular example the features we gave the network were audio recording of birds in the form of spectrograms in low resolution and the audio length.
A spectrogram is a picture that shows the magnitude of each frequency at a particular time window.

Code explantation

The code has a main function which contains a loop that runs for a specified number of iterations and within that loop we perform forward propagation, calculate, and save the cost then backward propagation and update the weights and biases.
The weights and biases are held in a dictionary with appropriately named keys that point to arrays containing the individual weights and biases.
To test the network we used two methods.
The first was saving the cost of the network with the training data during the main loop to an array and then graphing that array as a function of the iteration.
The second was generating a confusion matrix of the fully taught network with the testing data as input.
The cost graph helped see how many iterations the network took until it stopped having significant improvements.
The confusion matrix was useful for judging how good the network was at identifying the different bird species as well as finding the points of failure of the network.[1]

[1] Neural network code

The database

The database we used was extracted from the gbif.org site which contains a large collection of bird recordings from numerous different databases.
From the site we download .csv files which contain links to audio recordings or pictures as well as some auxiliary information.
After uploading the csv files to google sheets we removed some faulty occurrences as well as the ones which were not applicable for our use case and also removed the columns that didn't contain useful information for us.
We downloaded the modified google sheet as a .csv file and loaded it to a numpy python array.
We filtered the array in python to remove some more faulty occurrences[2].
Finally we ran a loop of the array, created a spectrogram from each occurrence, reduced the resolution of that spectrogram and saved it together with the audio length to another array.[3]
The reason we thought a spectrogram would work well for the network is that a spectrogram contains both the information about the timing of the bird calls (also contained in the .wav audio file) and information about the frequency of the bird calls (not obvious in the .wav audio file) as well as the fact that sounds dampen as they travel through air so observing different amplitudes for different bird species is unreliable but frequencies and timings dont change as they travel.
We saved that array as a .npy file for easy usage in training the network, we also saved an array of the targets for each occurrence (the index of the two arrays match between pairs of occurrence and target as well as match the audio data links in the .csv file).
After removing a few instances that failed to load during the spectrogram creation phase we finally had our data ready for use.[4]

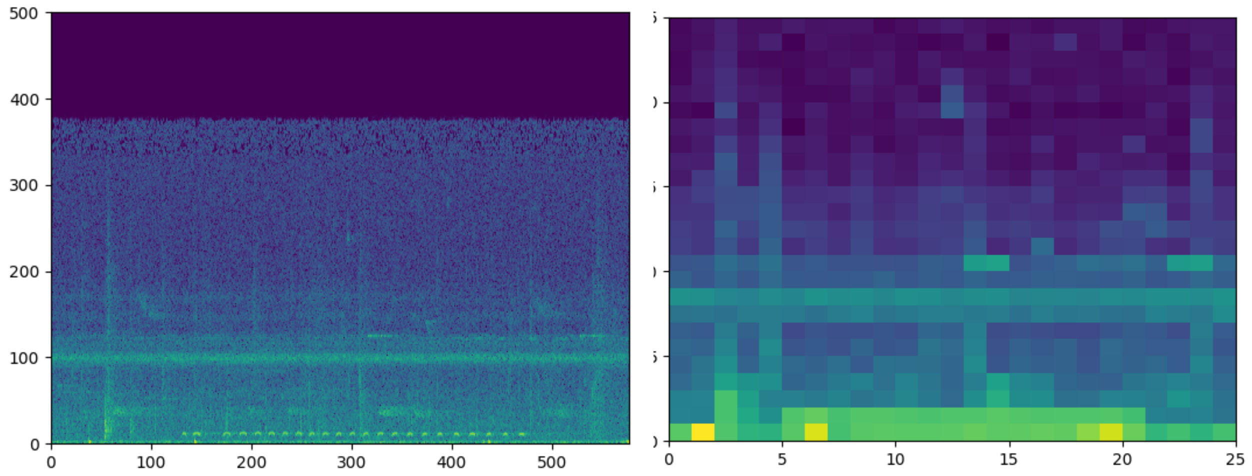
[2] Audio data links
[3] Data generation code
[4] Data, target, features

Comparison of spectrograms at different resolutions

The y axis is for frequency and the x axis for time.

Pigeon audio recording

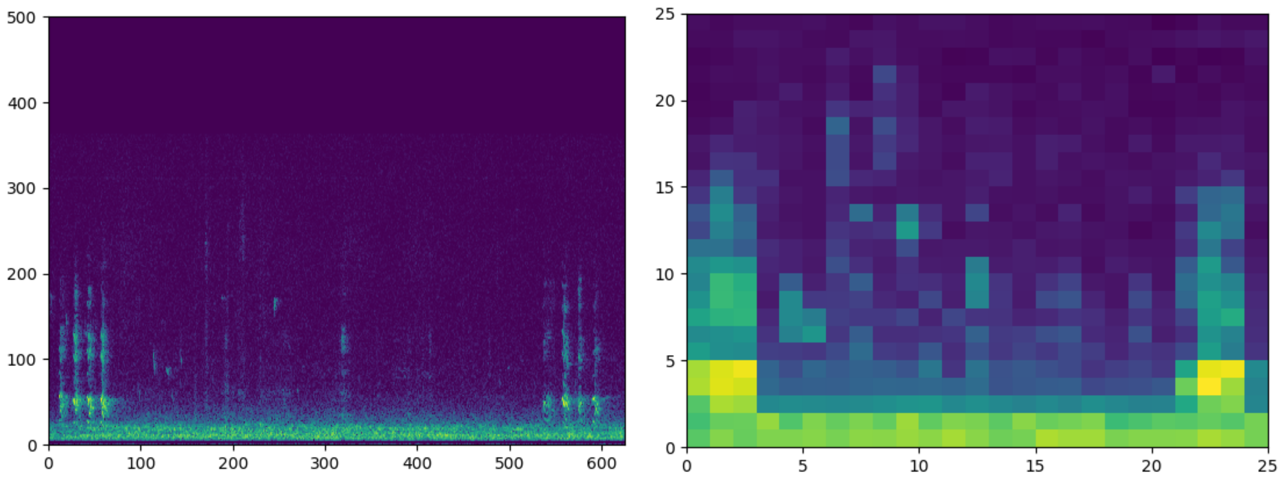
Spectrograms of a pigeon recording.



The pigeon calls in the left spectrogram are the small collection of bright yellow dots at the bottom of the spectrogram.
Finer details such as the distance between the pigeon calls and the hat shape of the calls themselves are lost, but the low frequency of the call is preserved.

Crow audio recording

Spectrograms of a crow recording.

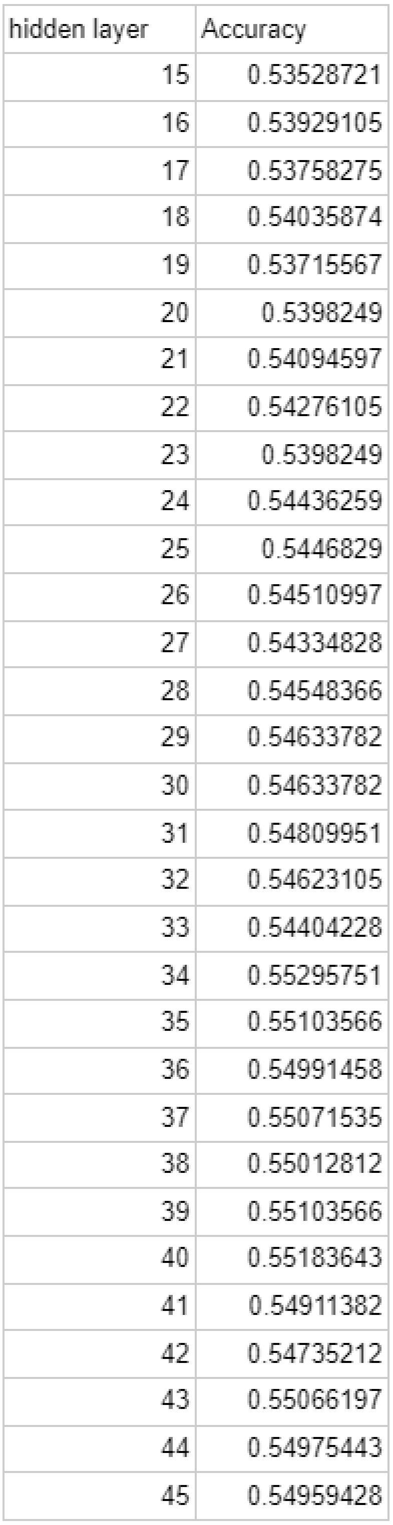


The vertical streaks at the start and at the end of the specogram are the crows calls.
The shape is mostly lost but the frequency is preserved and the time difference between the two different calls is also preserved.

Examples of the training and testing of networks

Training and testing networks for optimization

To find the best length for the networks hidden layer we used a brute force approach. We simply trained and tested 30 different networks with hidden layers ranging from 15 to 45 neurons.
The results can be seen on the right, and they show that a hidden layer larger then 24 neurons does not provide much benefit.
We went with a hidden layer of 25 neurons In our network.



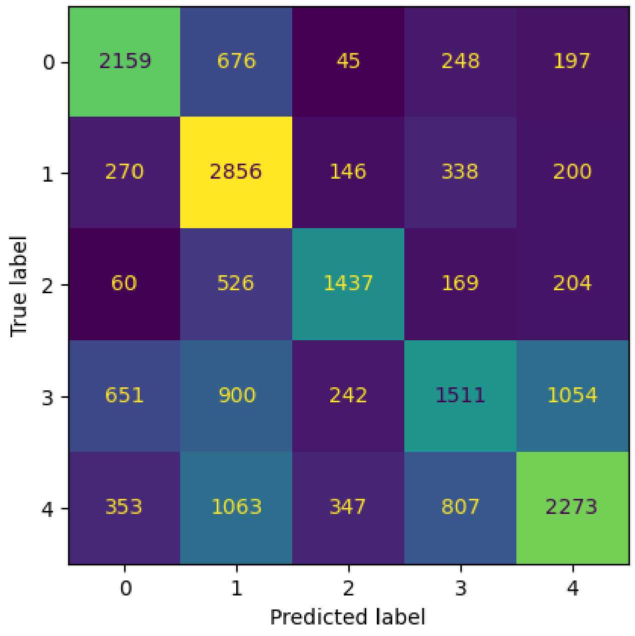
hidden layer	Accuracy
15	0.53528721
16	0.53929105
17	0.53758275
18	0.54035874
19	0.53715567
20	0.5398249
21	0.54094597
22	0.54276105
23	0.5398249
24	0.54436259
25	0.5446829
26	0.54510997
27	0.54334828
28	0.54548366
29	0.54633782
30	0.54633782
31	0.54809951
32	0.54623105
33	0.54404228
34	0.55295751
35	0.55103566
36	0.54991458
37	0.55071535
38	0.55012812
39	0.55103566
40	0.55183643
41	0.54911382
42	0.54735212
43	0.55066197
44	0.54975443
45	0.54959428

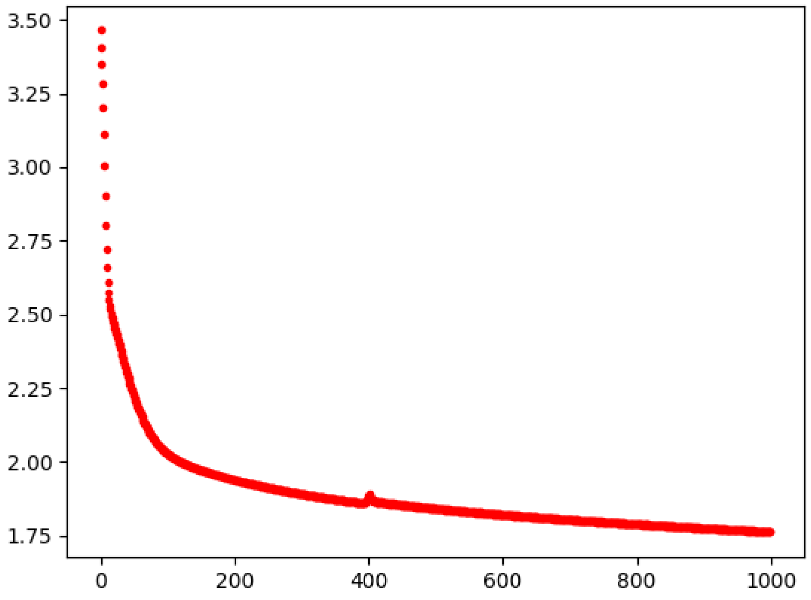
Chart for interpreting confusion matrices:

index	scientific name	common name	audio example
0	corvus	crow	crow_audio
1	columbiformes	pigeon/dove	pigeon_audio
2	passer	sparrow	sparrow_audio
3	rallidae	rail bird	rail_bird_audio
4	scolopacidae	sandpiper	sandpiper_audio

Training and testing networks for performance evaluation

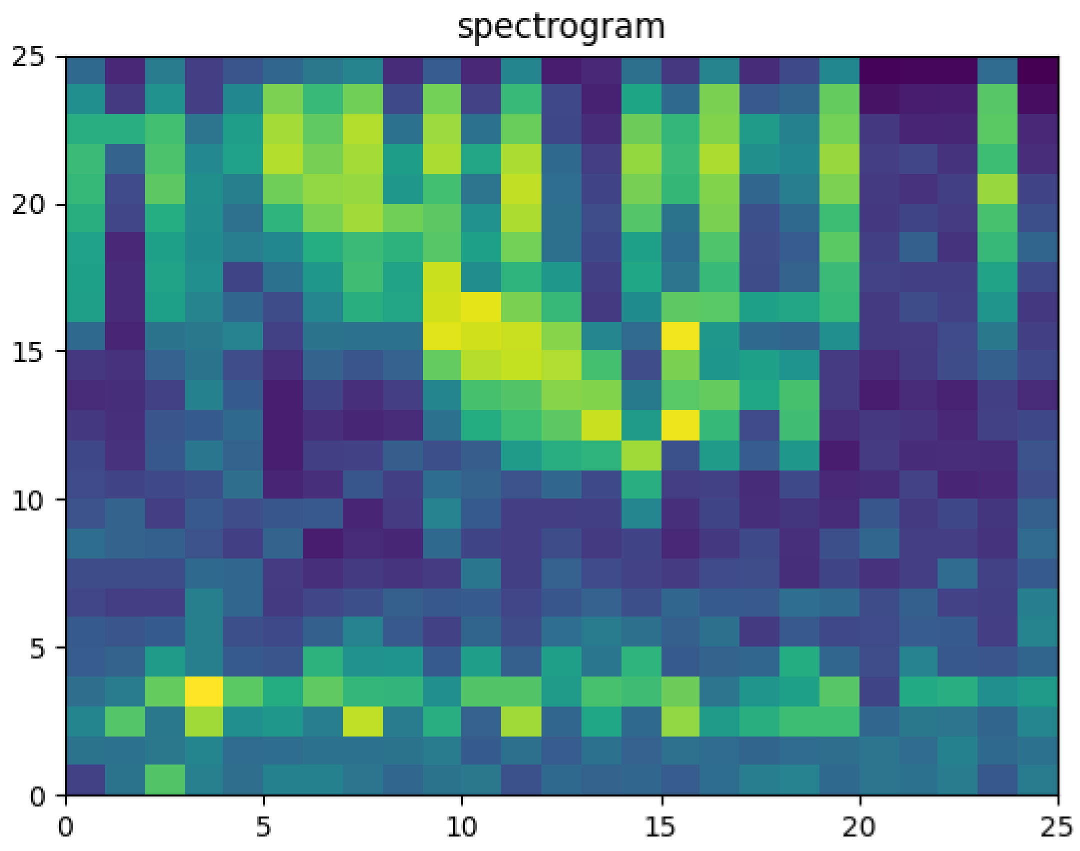
On the left we see confusion matrices where the columns are the label the network predicted and the rowes are the true label.
On the right we see graphs of the cost as a function of the iteration.





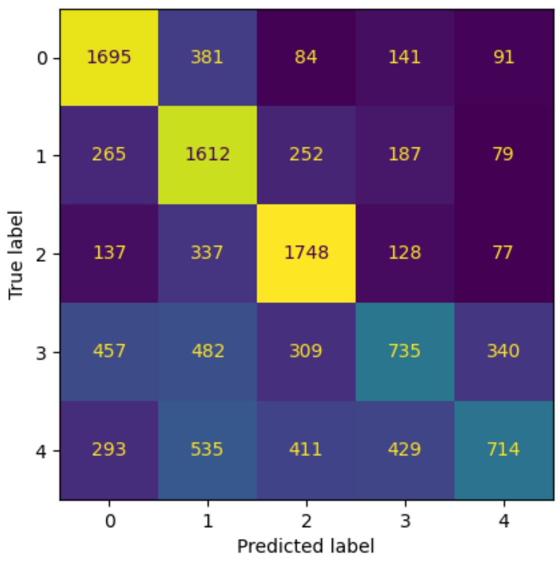
Accuracy = 55%
We can see that a network with one hidden layer of 25 neurons improves for about 1000 iterations after which there are no more significant improvements.
The network's main failures are with identifying rail birds and sandpipers.
It also tends to confuse all birds for pigeons.

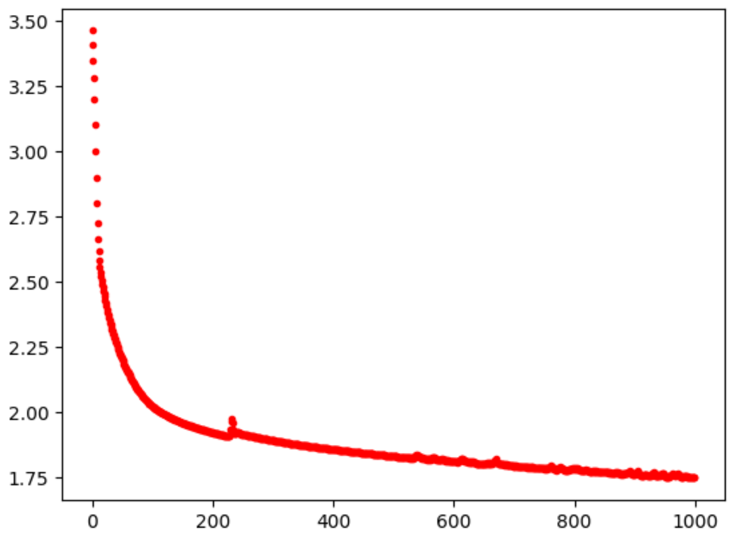
let's look at where the network fails the hardest (as measured by the cost function):



Audio recording
This particular example is of a pigeon and the network provided us with the following results for it:
Crow - 0.96472737
Pigeon - 0.09631146
Sparrow - 0.06179602
Rail bird - 0.58368133
Sandpiper - 0.91923393
(the closer the number is to one the more the network thinks this recording matches a particular species)
The network identified this recording as a crow, but we can also see that it gave a high likelihood of this example matching a sandpiper.
When we listen to the audio we can hear at the start of the recording the faraway call of a pigeon but we can also hear a much closer call of a different bird.
This might be the reason for the network's high confusion in this example.

Lets see how the network behaves when we feed it an equal number of examples from all species.

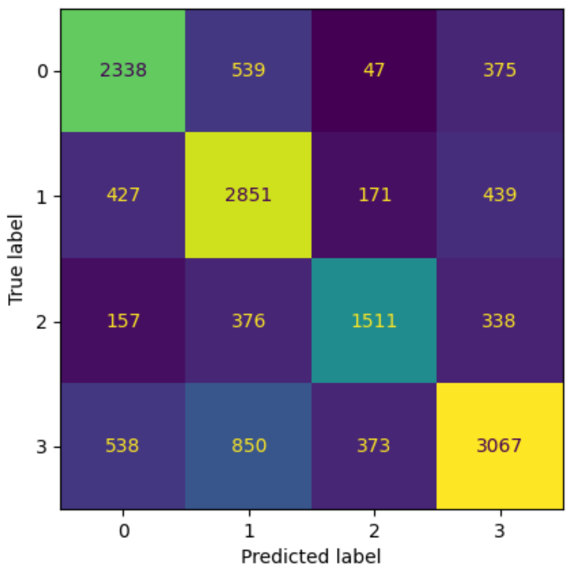


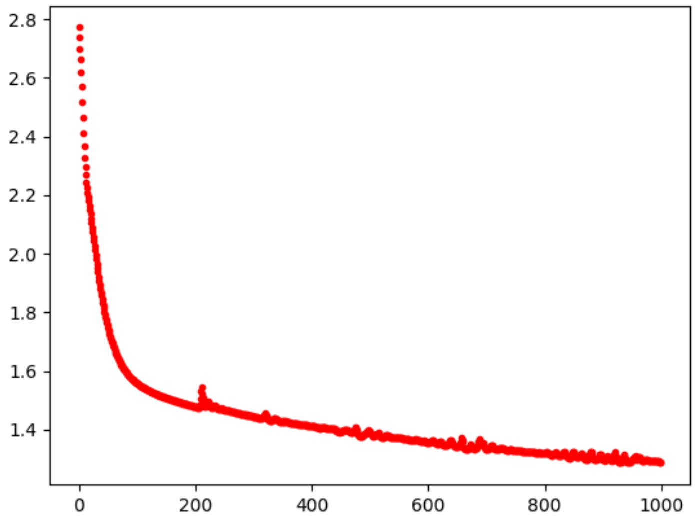


Accuracy = 55%

While the total accuracy did not change much we can see that the identification of species which had a large number of examples got worse and the identification of species which always had a small number of examples got better.

What about when we remove the most problematic species (the rail bird) to identify.



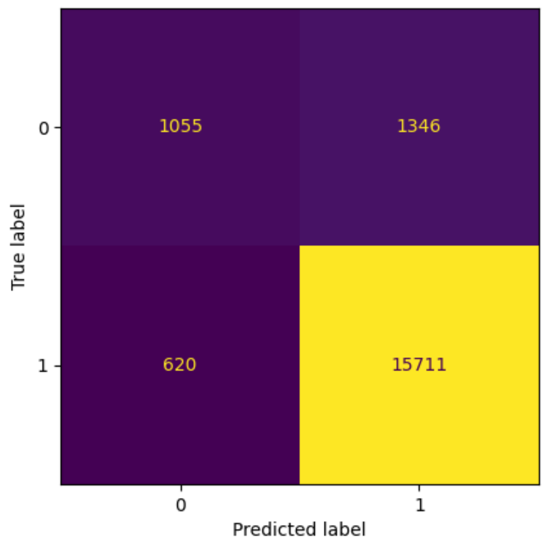


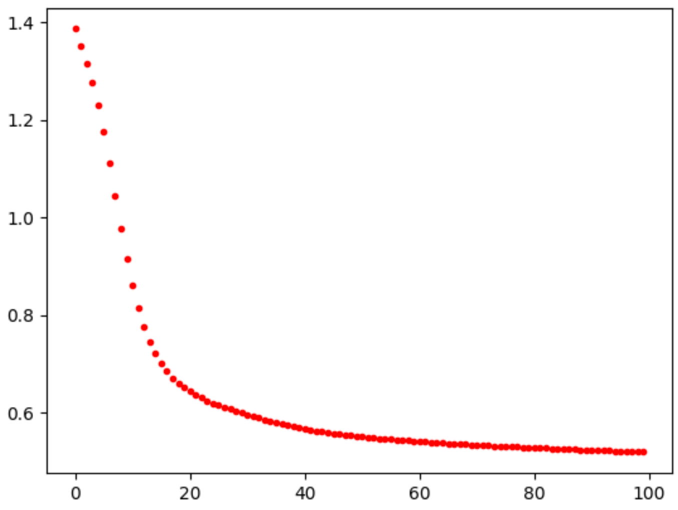
Accuracy = 68%

The highest accuracy is achieved when we have the network distinguish between one species and all the rest.

Sparrow or not a sparrow

Sparrow - 0



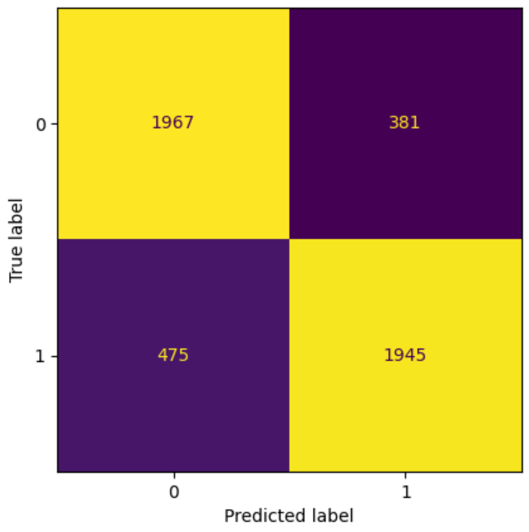


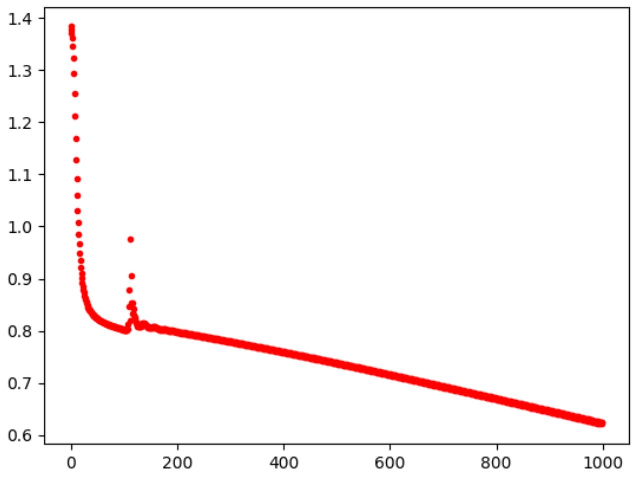
Accuracy = 90%

When we look at the confusion matrix we can see that the network can't really tell the difference between sparrows and the rest of the birds.

It is simply guessing that there isn't a sparrow and is usually right because there are fewer examples of sparrows than of the rest of the birds.

Lets see how it performs when the number of examples is equalized.





Accuracy = 82%

Still a very impressive performance considering we got rid of a large number of training examples (to compensate we increased the number of iterations).

Conclusions

- In all tests conducted the network performed better than guessing. This means that the network was truly able to learn the difference between the features we provided for different bird species.
    - When distinguishing between 5 different targets random guesses would be right about 20% of the time, the network was right 55% of the time.
    - For 4 different targets random guesses would be right about 25% of the time, the network was right 68% of the time.
    - For 2 different targets random guesses should be right about 50% of the time, the network was right 82% of the time.
- When we make all targets have an equal number of examples, We only see a slight dip in the accuracy of the network. This indicates to us that the main thing the network is learning about the different bird species is the differences in their calls and not their differing amounts of training examples.
    - For five targets (the target with the largest number of examples having 2.02 times more examples than the target with the smallest number of examples) we saw a dip in accuracy of less than 1% (54.64% before 54.57% after)
    - For two targets with a difference of 6.8 times we saw a dip in accuracy of 8% (90% before 82% after)
- When we compare the spectrograms as they are originally generated to the lower resolution ones we can see that most data about the shape of the bird call as it appears in the spectrogram is gone and all that remains is a dot at the frequency and time of the bird call. This leads us to believe that the patterns the network is looking for in the spectrogram are the frequencies of the bird calls and the time difference between adjacent bird calls.
    - The calls of a pigeon which are close together timewise and all happen at the same frequency get smushed together into one long line at that particular frequency. The hat shape of the calls is lost as well as the time difference between individual calls.

    - The calls of a crow appear as four close streaks going from a high frequency to a low but after we reduce the resolution they instead appear as one thick line.
- When we look at instances the network identified incorrectly we see that there are a mix of bird calls in the recording.
  This tells us that we might be able to improve the network further by not limiting each recording to only one target and by cleaning the database from recordings which don't match their target well.
    - We can hear the bird that matches the target but along with it we sometimes hear much more prominently the calls of other birds.

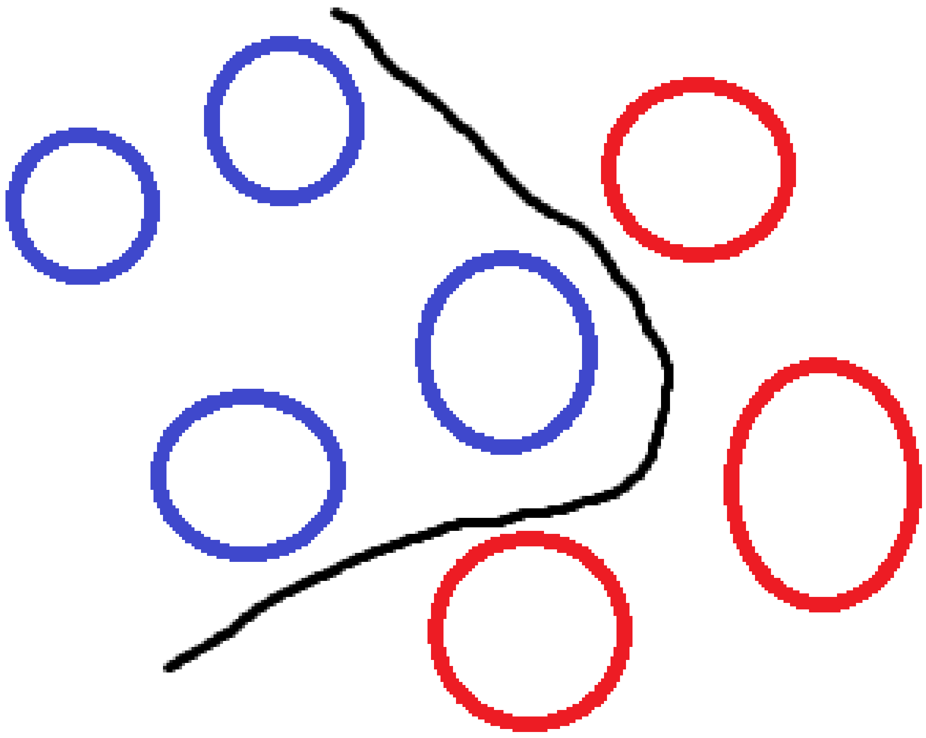
Part 2 - Centroid Method

In this part we will use scikit in order to produce centroids from the spectrogram and input the properties of the centroids as features to a neural network or a classifier model that is dependent on separation.

Theory

A centroid is the imaginary or real location representing the center of a cluster.
A cluster refers to a collection of similar data points.
An SVC model, or Support Vector Classification model, is an SVM model which applies a linear or nonlinear kernel function inorder to perform classification and it performs well with a large number of samples.
SVM, or Support Vector Machine, are supervised learning models which, with a corresponding learning algorithm, will perform classification and regression analysis. The explanation of how it does that is very complicated and involves complex high dimensionsional math, which is way above our level to explain.

A kernel or 'kernel trick' is a way of transforming data so it could be



linearly separated. i.e:

In these examples the data cannot be separated linearly, so in order to separate it linearly we must transform the data.

The transformation of the data is performed by what we call a kernel function. A kernel function will take a high dimensional input, apply a function to the inputs, and return the dot product of the inputs after the function transforms them.

The mathematical expression of the kernel function is:

$K(x, y) =< f(x), f(y) >$, where K denotes the kernel function, x, y are multi dimensional inputs, and f is a function that maps n dimensions to m dimensions.

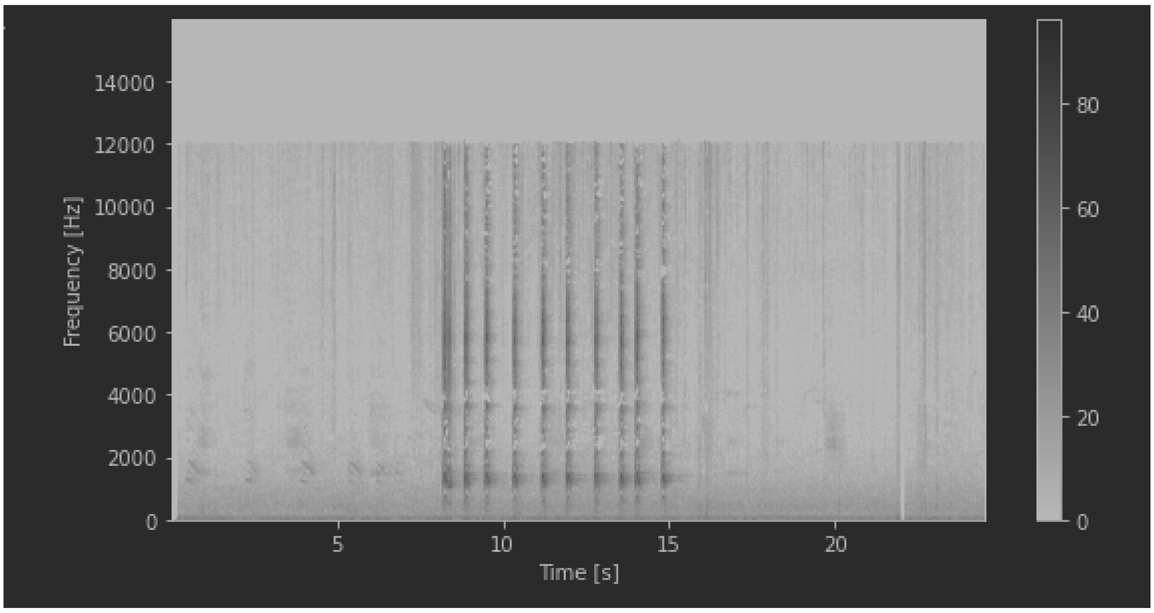
For example $let f(x) = (x_1 x_2 x_3)$ and $let x = (1, 2, 3), y = (4, 5, 6)$

Then

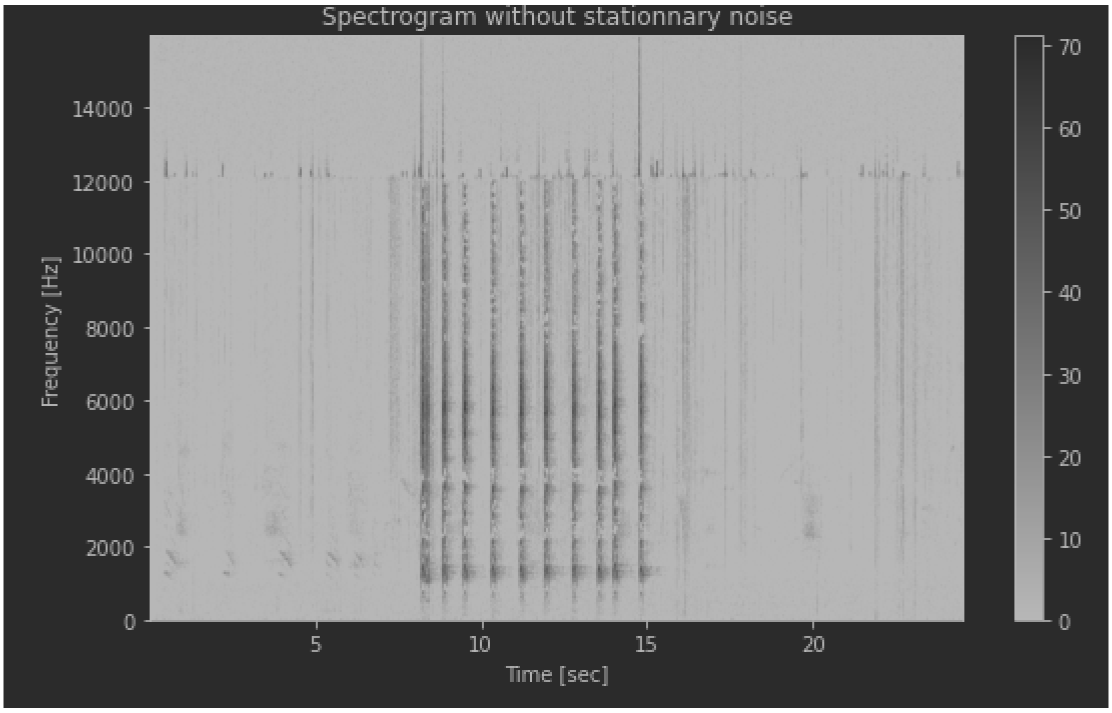
$K(x, y) = < f(x), f(y) > = < x!, y! > = < 1 * 2 * 3, 4 * 5 * 6 > = < 6, 120 > = 6 * 120 = 720$

Code Explanation

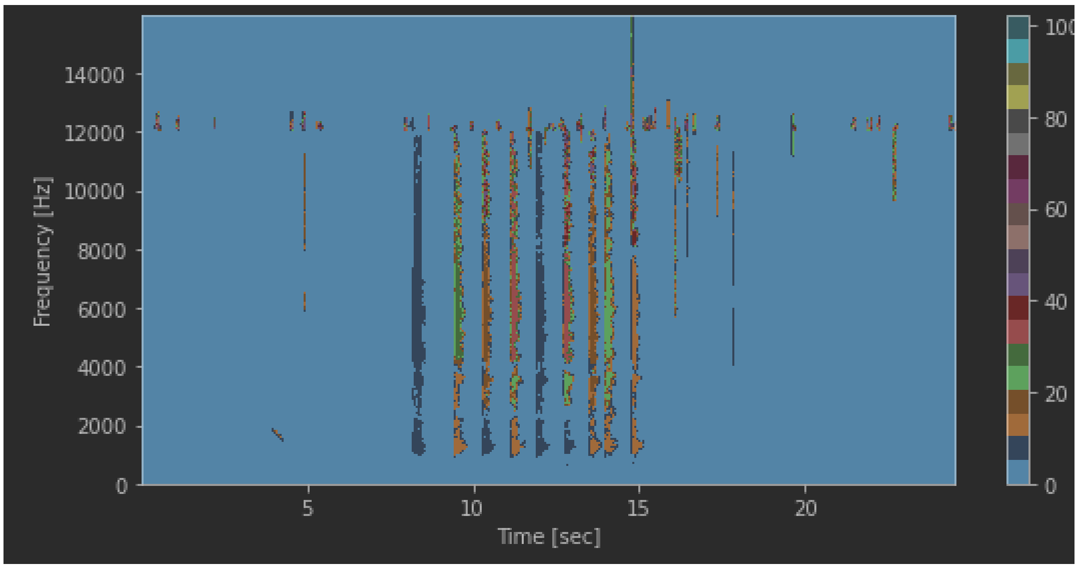
First we receive the audio files and transform it into a spectrogram like we did in the 1st method, but afterward we transform the spectrogram.



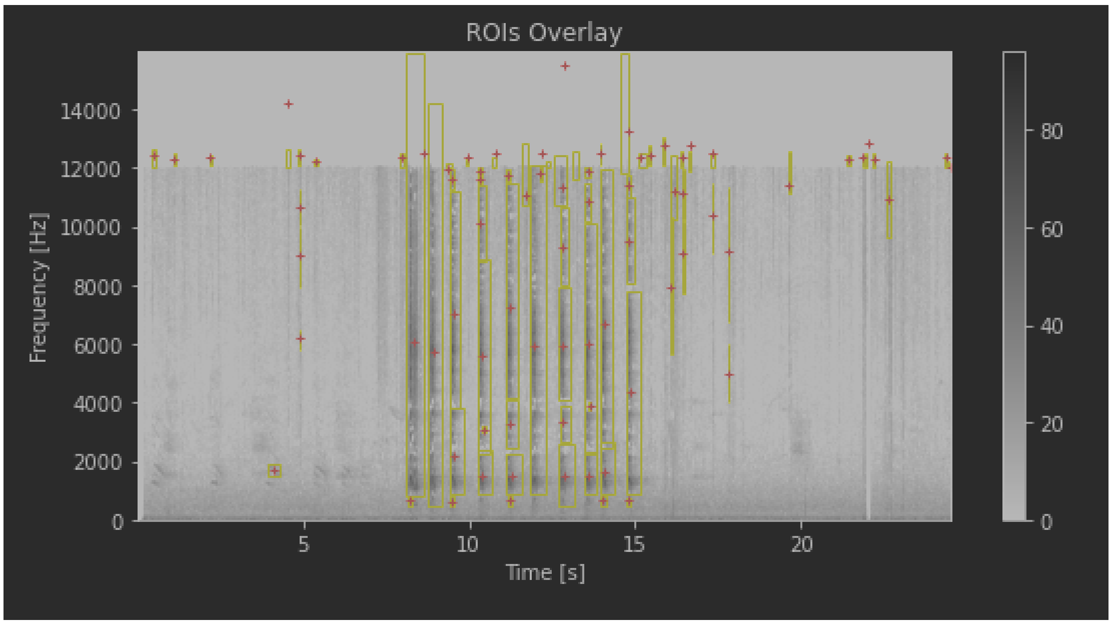
Now we reduce noise by specifying the baseline volume and applying a median equalizer.



Afterward we smoothen the spectrogram. Now, we will generate centroids in the spectrogram by using a mask on the spectrogram.



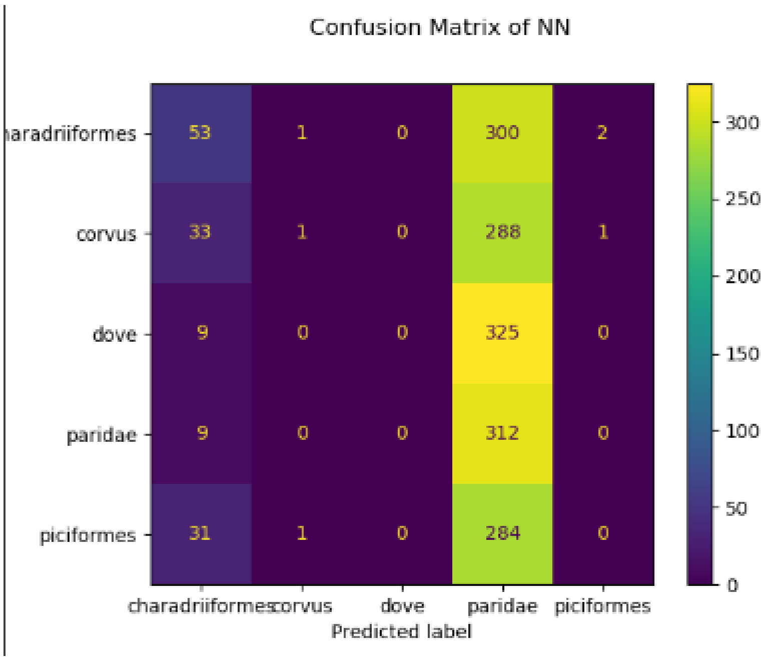
Mask applied on spectrogram



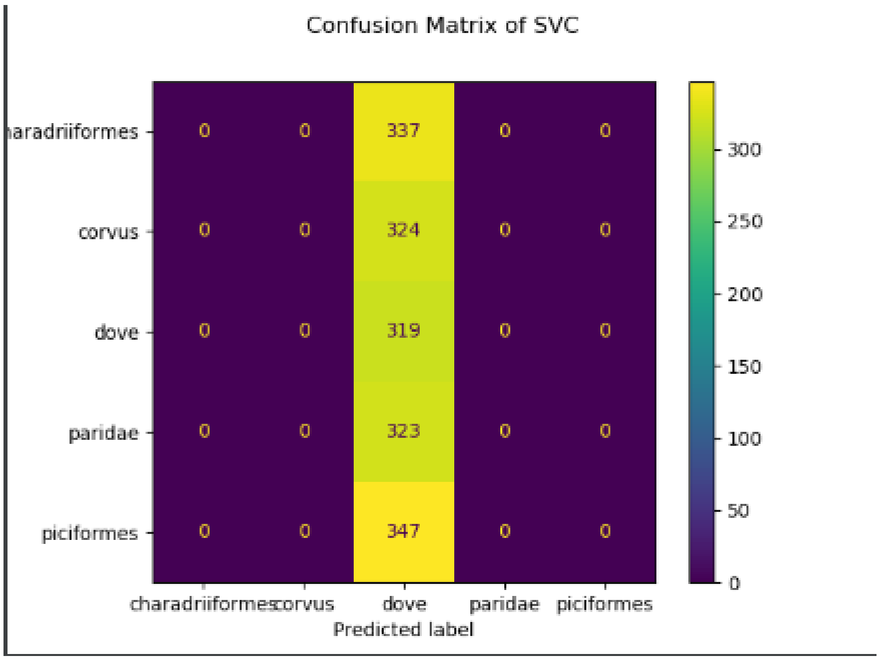
Selected Centroids

Now we can use the features of the centroid as inputs and the type of bird in the spectrogram as the label of the data. As any given spectrogram will have many centroids, it's very easy to gain many samples. For example, a given spectrogram, if it's long, can have around 400 centroids, and each of these is an example, now imagine having 50,000 spectrograms, that could be 1,000,000 samples easily. Though in this specific example, the program generated 77 centroids from a 25 second recording. That means we can add a lot more features to the model, but this can be done in further research (for example taking the delta_t of the spectrogram, and delta_f of the spectrogram, and doing convolution and photo recognition on it, as the main problem of photo recognition is the huge amounts of samples needed). When we generate the centroids, we hope that they will represent the calls of the given bird in the recording. The Features used from the centroids are centroid_f, centroid_t, duration_t, bandwith_f and area_tf (t in units of seconds and f in units of hertz). Centroid_f and Centroid_t give us the baseline of the centroid while duration_t and bandwith_f give us the timespan and frequency span and area_tf will give us the area covered by the centroid. You may see in the diagrams above that the program generates more than one centroid for a bird call, we hope it will average out over a large number of samples as it does that for every sample.
After generating the dataset we feed the data to a neural network and an SVC model, in order to find out which model is better for our task.

When inputting to the NN we normalize the data because if we don't, a single output will become dominant:



And for the SVC we don't normalize the data, because the same will happen:

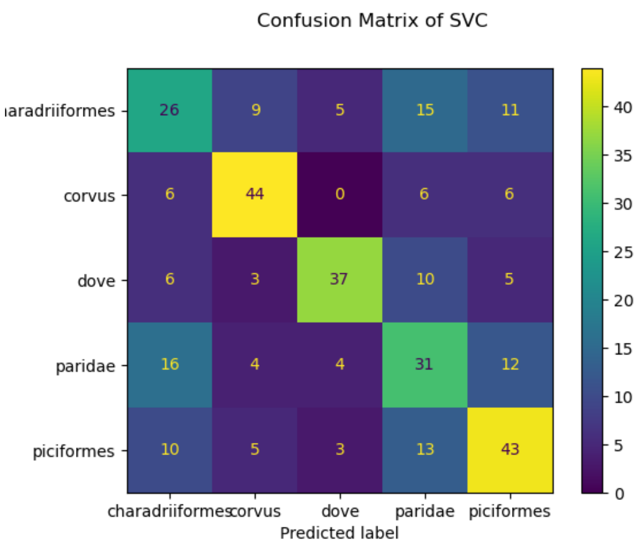


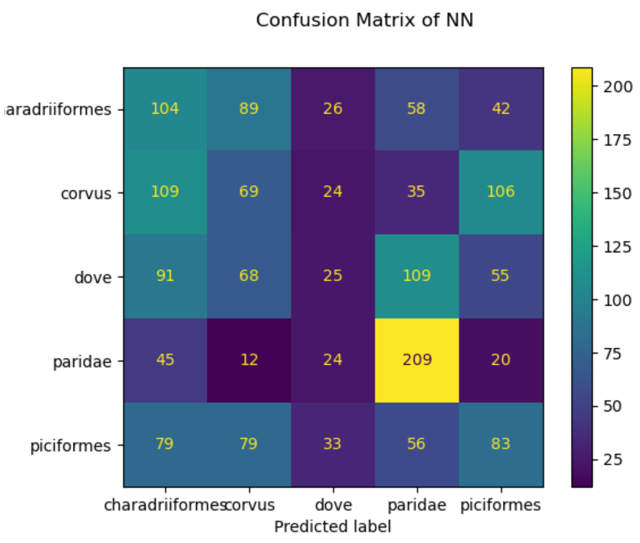
The NN has two hidden layers, first of size 16 and second of size 8, as I found that these sizes work best.

Examples of training and testing of networks

First we evaluate both models with varying sample sizes.

1000 samples:



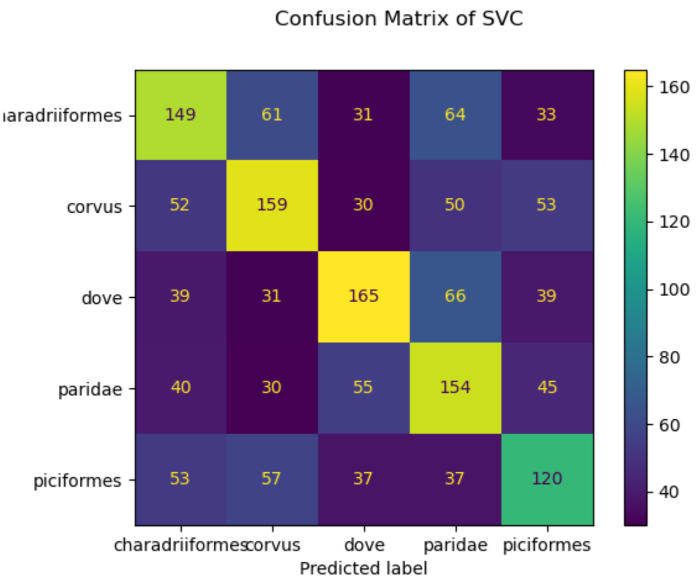


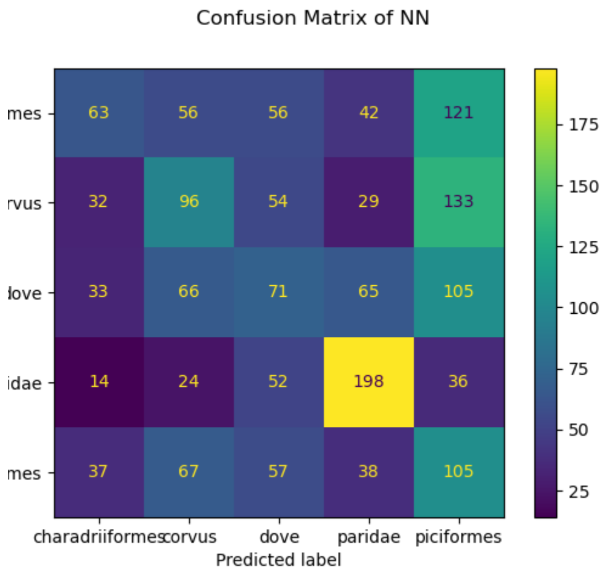
Accuracy of SVC: 0.54

Accuracy of NN: 0.44

We can see that for this low number of samples both models manage to be better than random, that's a good sign, and we can see that the SVC model pulls ahead of the NN by accuracy.

5000 samples:



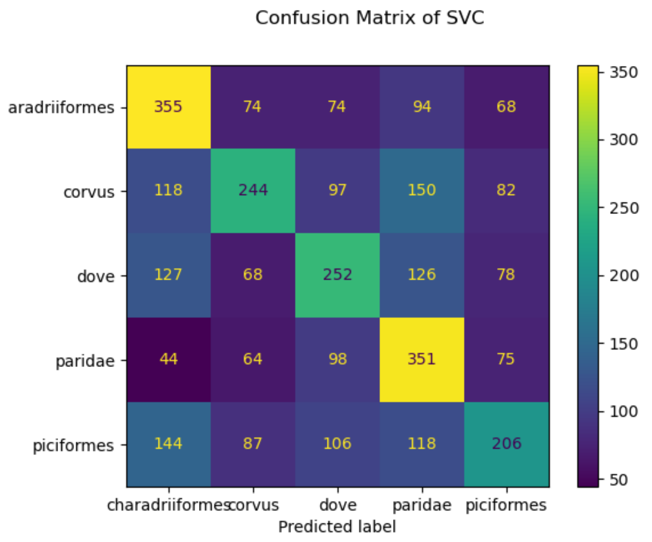


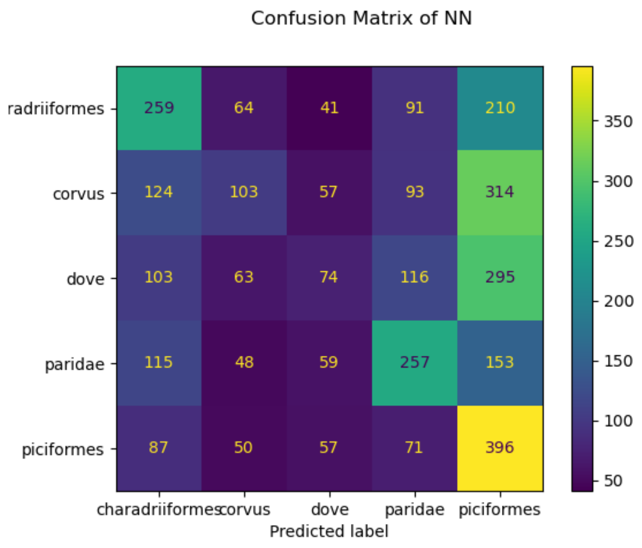
Accuracy of SVC: 0.45

Accuracy of NN: 0.32

By raising the number of samples, the accuracy of both models went down, though we can see that at this number of samples, the NN starts to have a dominant output - the piciformes output is starting to dominate the network.

10000 samples:





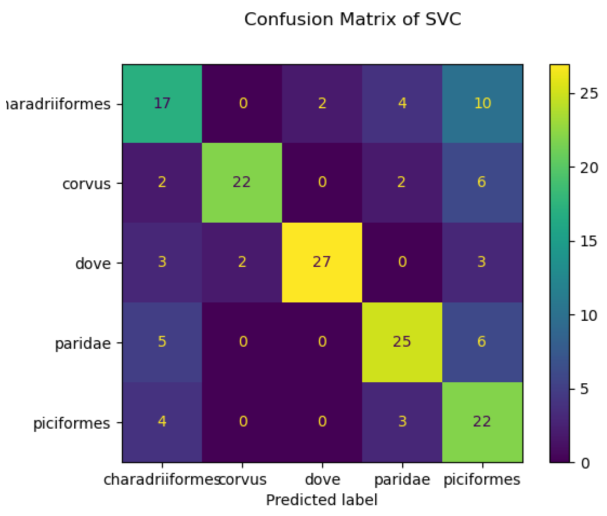
Accuracy of SVC: 0.43

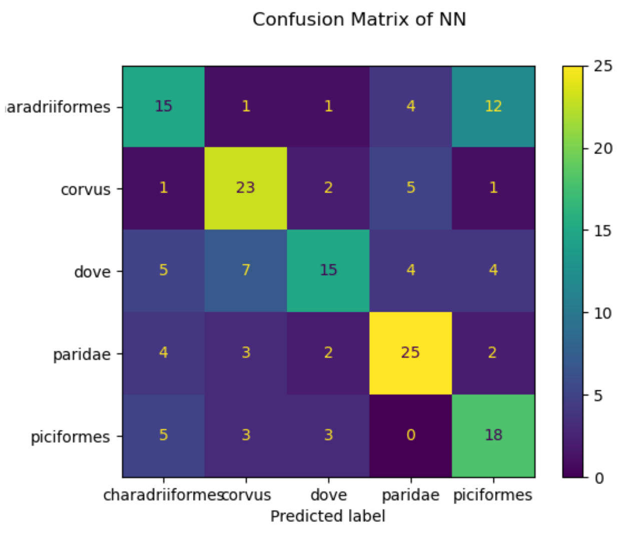
Accuracy of NN: 0.33

We see the same trend when increasing the samples from 1000 to 5000, accuracy went down and piciformes dominate the NN.

Let's try to lower the amount of samples and see the effect on the model.

500 samples:





Accuracy of SVC: 0.68
Accuracy of NN: 0.58

Lowering the sample amount to 500 made the models better, though we see the same trend over all sample amounts, it seems the NN is 10 percent worse than the SVC model for the task.

Lowering even further:

When we lower to 250 samples, the accuracy of both models go down, that means that the accuracy maximum is in between 250 and 500 samples.

Conclusions

We have shown that a model based on centroid features for bird call recognition is viable while using an SVM model is superior to a NN model.
We have shown that the accuracy of the models falls considerably as we increase the number of samples after a certain point.
The cause for that might be that certain samples confuse the models and make them worse and when we take a lower amount of samples from the spectrograms we manage to only take the significant centroids and don't add useless noise this might also be a problem of over training even though we are not increasing the number of iterations but the number of samples, because we extract a number of samples from each spectrogram there can still be a case of overtraining for specific examples when we increase the number of samples.
As we have so many samples and not much to do with them (see last point) it might be a good idea in further research to use the features gained from the centroids for more complicated models.

Addendums

Footnotes:

1) Neural network code (first part)
2) Audio data links (first part)
3) Data generation code (first part)
4) Data, target, features (first part)
5) Data generation (second part)
6) Generating Features (second part)
7) Training and Testing the Models (second part)
8) Inputs, Labels (second part) (with 250 samples, if interested in more request and we will send)

Bibliography:

Python packages:

1) Scikit-learn: Machine Learning in Python
2) API design for machine learning software: experiences from the scikit-learn project
3) Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. (Publisher link).
4) J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.
5) Robert, J., Webbie, M., & others. (2018). *Pydub*. GitHub. Retrieved from http://pydub.com/
6) Chandra, R. V., & Varanasi, B. S. (2015). *Python requests essentials*. Packt Publishing Ltd.
7) Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Database:

1) GBIF.org (23 June 2022) GBIF Occurrence Download https://doi.org/10.15468/dl.mfepwq
2) GBIF.org (23 June 2022) GBIF Occurrence Download https://doi.org/10.15468/dl.htjnpf
3) GBIF.org (23 June 2022) GBIF Occurrence Download https://doi.org/10.15468/dl.8xcu6x
4) GBIF.org (23 June 2022) GBIF Occurrence Download https://doi.org/10.15468/dl.w3yk3c
5) GBIF.org (23 June 2022) GBIF Occurrence Download https://doi.org/10.15468/dl.gnpuhx