Final Year Project Report: Final Submission

Full Unit – Final Report

Chord Recognition and Predictions via Sound Classification

Shaw Chifamba

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Li Zhang

Department of Computer Science

Royal Holloway, University of London

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count:

Student Name: Shaw Chifamba

Date of Submission:

Signature: Shaw Brandon Chifamba

Table of Contents

Table of Contents

Chapter 1: **Aims, Objectives and Motivations**

1.1 Motivations

As a guitarist and musician, understanding the notes and chords you are playing is vital to not only compose but to play along to music. This becomes a bigger issue when variations of guitar tunings are used as chord shapes become altered thus you can no longer rely on memory to identify a chord name.
As an example, standard tuning is EADGBE, and a popular alternative tuning is DADGAD. As all guitarists should be taught in standard tuning the regular patterns and shapes should be memorised and given a selection of fret numbers, able to deduce a chord name. Diagrams can be found in the appendix. Figure 1 Figure 2

I wished for a system that was able to listen to your guitar and output a chord name regardless of guitar tuning, so I decided to research into the topic and experiment with machine learning algorithms for both audio and images to understand how feasible the task is. Additionally, I also wanted to understand why systems like this are not readily available for public download – this excludes certain websites that implement this via YouTube songs such as: https://chordai.net/ , https://chordify.net/ and https://chordu.com/

1.2 Original Aims & Objectives

As my project is an original idea, I had 2 very basic original aims:

- To detect live guitar chords and recommend chord progressions.
- To understand machine learning algorithms for sound classification.

My final deliverables regarding project were as follows:

1. The program must have a full object-oriented design.
2. The program will have a GUI that will be used to develop the sequences of chords and potential chords.
3. The program needs to be able to recognise chords directly from an audio input.
4. The program needs to be able to use previous classifications to determine the best next chords.
5. The program should be as efficient as possible.
6. Full report
    a. Detailed explanation of techniques used on the project.

b. Discuss the choice of sound classification algorithm.

c. Evaluate the accuracy of machine learning algorithm/s used.

d. Evaluate results alongside issues faced and tackled.

e. Include documentation of the code alongside comments and justifications.

f. Review of the literature used to generate final program.

g. Evaluation of accuracy of algorithms used via a variety of researched techniques.

I will provide an analysis into why some deliverables were met and why some were not in both my evaluation of machine learning and self-evaluation.

1.3 Evolved Aims & Objectives

After researching and producing my interim reports which can be found in 2.1 and 2.4 my aims and objectives became the following:

Aims:

1. Develop a broad understanding of the research methods, audio processing algorithms, machine and deep learning techniques and libraries commonly used to classify audio and what approaches have been taken in chord recognition.

2. Understanding how TensorFlow and Keras can be used alongside transfer learning, including reporting loss, accuracy, and validation loss.

Objectives:

1. To compare different machine learning and deep learning algorithms on different representations of audio.

2. To compare different audio processing techniques for best accuracy and speed when used in conjunction with machine learning.

3. To investigate and evaluate the feasibility of machine learning and audio processing techniques on limited audio datasets.

4. To evaluate and determine suitable error rates for machine learning guided sound analysis.

5. To assess ability to create a real time automatic chord recognition system based on complexity of machine learning approach (from machine learning standpoint).

6. To compare different machine learning and deep learning algorithms implementations in Python on different representations of audio.

7. To investigate and evaluate the feasibility of selected machine learning techniques using a small proof-of-concept and image recognition tasks – related to audio.

8. To assess ability to create a real time automatic chord recognition system based on complexity of machine learning approach (from chord recognition standpoint).

1.4 Future Career Prospects

The entire project has focused on data collection, data cleaning, algorithm creation, modelling and evaluation, along with an in-depth analysis of popular machine learning frameworks and tools such as TensorFlow, Pytorch, Google Colaboratory and Jupyter Notebook.

Currently, I hold an offer to study Precision Cancer Medicine at Oxford University where the skills I have gained to apply a domain [music] to the area of machine learning will greatly help me apply machine learning techniques to the field of cancer research. In addition, the use of Python is greatly appreciated in the realm of bioinformatics and data analysis, further providing me with a more computational foundation for my future research and career.

Chapter 2: **Theory & Literature**

2.1 Introduction to Transfer Learning

This section will build upon my interim reports and provide background information into chord recognition and sound recognition.

In both early reports we observe how image classification algorithms such as, ConvNets, HMMs, k-means clustering can all be used to attempt sound recognition. However, for my final deliverable my focus shifted towards transfer learning following discussions with my supervisor for more up to date machine learning methods towards sounds classification.

Transfer learning is process by which we take features trained and learned on one problem for instance an image classification task conducted on a large scale dataset and fine tune our new network using these features to learn a new task[1] Transfer learning is the preferred approach for my problem as we do not have the luxury of vast amounts of labelled chord data in order to train a network from scratch, an issue that I discuss in more detail on 5.1

The typical workflow is as follows:[1]

1. Obtain the base model you wish to use and load its pre-trained weights.
2. Freeze all layers apart from the output layers of the base model.
    a. Alternatively – the method I opted for, obtain the outputs of running your data through the base model [feature extraction]
3. Create a new model on top of the output for the base model.
    a. Alternatively – the method I opted for, use extracted features for a smaller model./
4. Train new model on the new dataset.

Keras notes that using feature extraction may be a disadvantage due to the lack of data augmentation that can be applied during each epoch. Data augmentation is an important step when training neural networks that is often seen in image recognition tasks, it is the process of changing features of the input. In the example of image recognition this can involve tasks such as rotating, inversing, stretching images so the model does not overfit itself, generalising on its training data as opposed to the wider nuances of the dataset.[2]

[1] https://keras.io/guides/transfer_learning/

[2] https://www.tensorflow.org/tutorials/images/data_augmentation

2.2 Interim Deliverables

In sections 2.3 and 2.4 you will find my interim deliverables. They cover a very broad range of machine learning algorithms with a very basic and rudimentary image classification example to support them.

They were heavily focused on identifying common machine learning algorithms for sound classification and covered CNNs, hidden Markov models and N-gram modelling. All methods discussed are plausible, however, they are quite tedious and not completely state of the art. This is where the switch to transfer learning started as it was easier to leverage existing CNNs than create models based on the algorithms below from scratch, especially with a lack of good labelled data.

This is not to say my research was incorrect as I indeed use some ideas present in both reports in my final deliverable such as, Log Mel Spectrograms and CNNs.

Transfer learning was introduced in chapter 2.1.

2.3 Early Deliverable 1 – Analysis on Different Machine Learning Algorithms Regarding Sound Classification

2.3.1 Aims

Machine and Deep learning algorithms such as, Convolutional Neural Nets (ConvNets)(CNNs), N-gram modelling, hidden Markov models, transfer learning, k-means clustering are used to attempt automatic chord recognition from guitar audio and in small cases full length songs. The aim of deliverable 1 was to develop a broad understanding of the research methods, audio processing algorithms, machine and deep learning techniques and libraries commonly used to tackle the problem to incorporate them into my proof-of-concept, and subsequently my final product.

Language processing was also investigated due to the similarity between natural language and chord progressions, this is in line with my 4th final deliverable.

“

1. The program needs to be able to use previous classifications to determine the best next chords.

”

2.3.2 Objectives

1. To compare different machine learning and deep learning algorithms on different representations of audio

2. To compare different audio processing techniques for best accuracy and speed when used in conjunction with machine learning

3. To investigate and evaluate the feasibility of machine learning and audio processing techniques on limited audio datasets

4. To evaluate and determine suitable error rates for machine learning guided sound analysis

5. To assess ability to create a real time automatic chord recognition system based on complexity of machine learning approach (from machine learning standpoint)

2.3.3 Abstract

In this report, I broadly review machine and deep learning algorithms that are relevant to the problem of automatic chord recognition and offer my interpretations. To this end, I will focus on the algorithms independent of how they are used in chord recognition tasks and focus primarily on why they are selected and their suitability to different sound representations.

Most techniques I have researched utilise convolutional neural networks, hidden Markov models, multitask and transfer learning and support vector machines. These algorithms are then applied in tandem with sound processing techniques such as the Fourier transform (including variations such as fast Fourier transform) and Harmonic Constant-Q transform, to produce representations of sound. Such representations of sound include harmonic pitch class profiles, Mel-frequency cepstral coefficients, Mel-spectrogram and more.

Keywords: Convolutional Neural Networks, Hidden Markov Models, Multitask Learning, Transfer Learning, Supervised Learning, Machine learning, Deep learning, Support Vector Machines, Fourier Transform, Constant Q-Transform, Harmonic Pitch Class Profile, Mel-Frequency Cepstral Coefficients, Mel-Spectrogram, Natural Language Processing, N-gram

2.3.4 Summary of Research

As stated in my abstract there are various audio processing and machine learning techniques that have been used in automatic chord recognition. My aim in this deliverable is to conduct research on the induvial algorithms and the most accessible way of programming them in Python as this is the language I will be using. Librosa [1] is a useful tool I will be using to create chroma features as it provides the tools with relative ease and existing works [2] also use Librosa.

Most of my discussion from this point on will be focused on Meinard Mullers Fundamentals of Music Processing with other existing literature to support any points made.

2.3.5 Template and Pattern Matching Approach

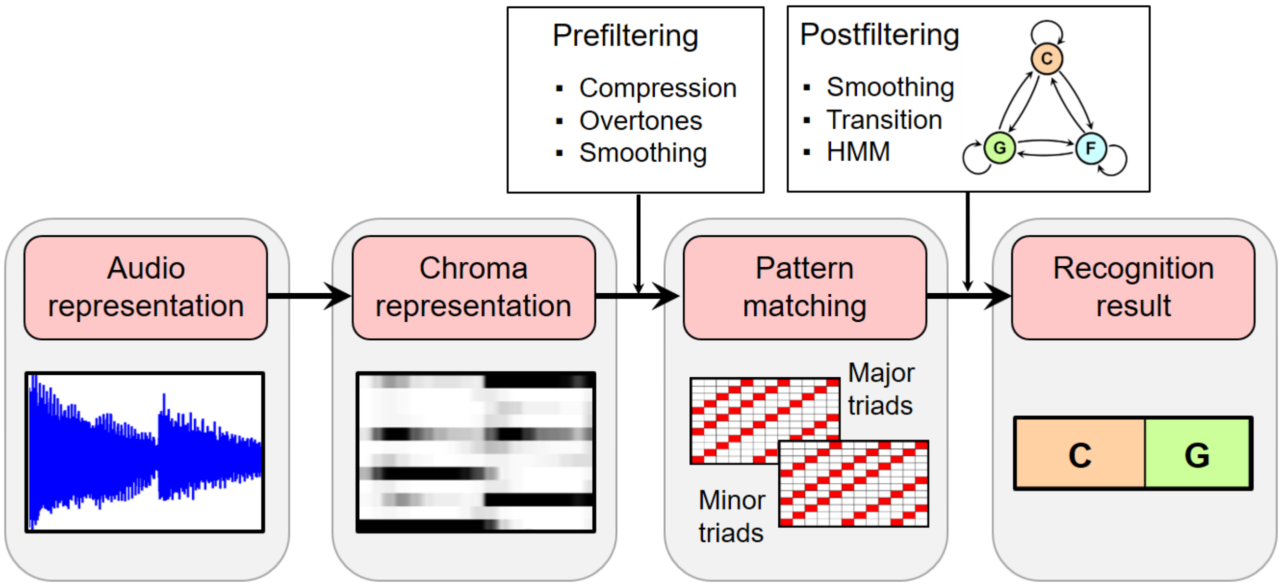


Figure 5.13 from [Müller, FMP, Springer 2015]

We see in the above diagram that there is a fundamental approach to starting this problem. We first must obtain our audio representation and transform it into a chroma-based feature representation. Many different authors have approached this differently, Humphrey & Bello [3], Bello & Pickens [4], and Cho et al [5] discuss computing the CQT as a chroma representation. Whereas Sheh & Ellis, Lee & Slaney, Cheng et al and others have used PCPs or HPCPs (Harmonic Pitch Class Profiles) which aim to preserve tonality [6].

In librosa we can create different chroma features as follows:

Fourier Transform based chroma features – librosa.feature.chroma_stft

Constant Q-Transform based chroma features – librosa.feature.chroma_cqt

The chromagram for each audio sample is produced by first loading the wav file into librosa, then passing the file into one of two of the functions listed above. You must pass a window size and hop size which are automatically set at 4096 and 2048 respectively.

Window size is the amount of time over which a waveform is sampled, in other words splitting the input into temporal segments [7]–[9], therefore our time record is 0.18s (4096 * 1/22050)[9] and subsequently the lowest frequency we can analyse is approximately 10hz which suits our task as the lowest E string on a guitar is approximately 83Hz[10] (assuming standard tuning).

Hop size is the number of samples between successive windows to give us our whole signal representation.

Muller then introduces template-based pattern matching, whereby the process is:

1. Develop a Chroma sequence X

2. Obtain set Y of possible chord labels

3. Precompute a set of templates Z such that each template represents a chord

4. Create a similarity measure that allows us to compare different chroma features

5. Create an algorithm that returns the maximum value of similarity between the set of templates Z and computed chroma sequence X and assign chord label Y

As stated in deliverable 2, there are many different types of chords that are selected for Y, with Bello & Pickens [4] being an example as they classified major and minor sevenths with their triads. This approach helps with simplicity and classification accuracy but ultimately falls short from a musical perspective. In addition, Mullers abstraction of only using 12 major and minor triads falls under the same criticism, although Muller does state this and claims it "convenient and instructive".

Each chord can be represented in a 12-bin vector, in the example of a C major we obtain ZC = (1,0,0,0,1,0,0,1,0,0,0,0) – on the assumption that octave shifts and notes with enharmonic equivalence (for example, G# and Ab) are considered. In addition, due to how transposition works we can shift semitones of the major triads to easily obtain our minor triad bin vectors.

2.3.6 Machine Learning Approach

I have covered a variety of machine learning approaches in deliverable 2, so instead I will focus on the machine learning techniques themselves and what they entail in this deliverable. I will briefly cover HMMs and CNNs.

Supervised learning was the focus of my research as unsupervised learning currently is out of my scope. The idea in principle is that the training data would be chroma features with their corresponding labels and the input data would be the audio we wish to classify. The output would be the predicted chord label.

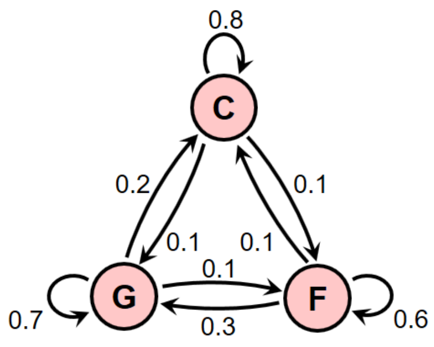
2.3.7 Hidden Markov Models

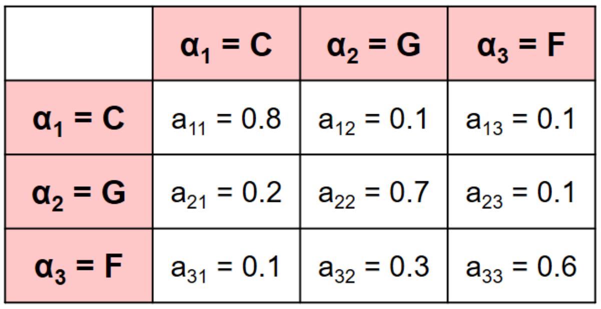
The following segment mainly follows Mullers notebook as mentioned before. Any other sources will be accredited.

In the realm of music theory, chord progressions have a statistical probability of being viable, for example {C, F, G} is a very common chord progression in C major. {C, F, D#} however, is not and would likely not occur in popular western music. We can use Markov chains to reflect this property.

A Markov chain is a model that describes the probability of a new state determined only by the state we are currently present at. Transitions that move according to probabilistic rules.[2]

Assuming we had a set A := {X1, X2, …., Xn} which had all the chord types we can have. At any N instance the change from one state in the set to another is derived from a set of probabilities associated with each state and every other state. This probability of change is only dependent on the current state and not the ones before it – otherwise known as the Markov property. The coefficients of each state moving to another are also independent and are called state transition probabilities.





	α_1 = C	α_2 = G	α_3 = F
α_1 = C	$a_{11} = 0.8$	$a_{12} = 0.1$	$a_{13} = 0.1$
α_2 = G	$a_{21} = 0.2$	$a_{22} = 0.7$	$a_{23} = 0.1$
α_3 = F	$a_{31} = 0.1$	$a_{32} = 0.3$	$a_{33} = 0.6$

Figure 5.24 from [Müller, FMP, Springer 2015]

As shown in this diagram the probability of staying within a singular chord is high but each chord has different probabilities of transitioning into different states. To do this initial state probabilities are given such that the model has an idea of how to start.

In this textbook Muller does not observe a sequence of chord types but rather a sequence of chroma vectors. Recall, that Cheng et al. [6] – in deliverable 2 - described a method of combining N-gram modelling with HMMs and chord types to predict chord sequences. Their paper may prove useful later in my project when I wish to predict chords based on emotion.

For Mullers system, the aim is recognition, such that HMMs are used to represent the idea of relation between chroma features and the states (chord types), using a probability model. Each state has a probability function that tells us about the likelihood for a chord to produce a chroma feature.

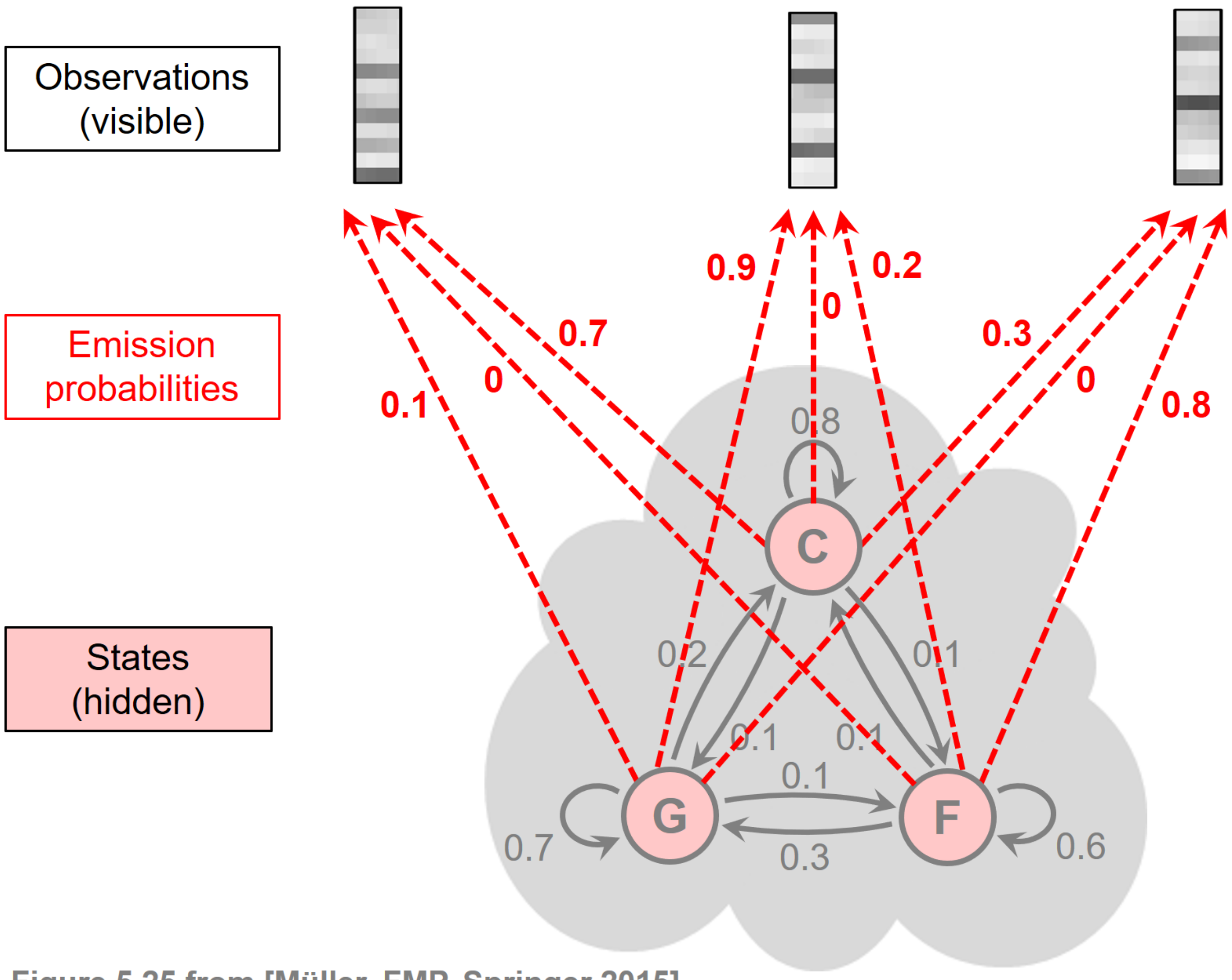
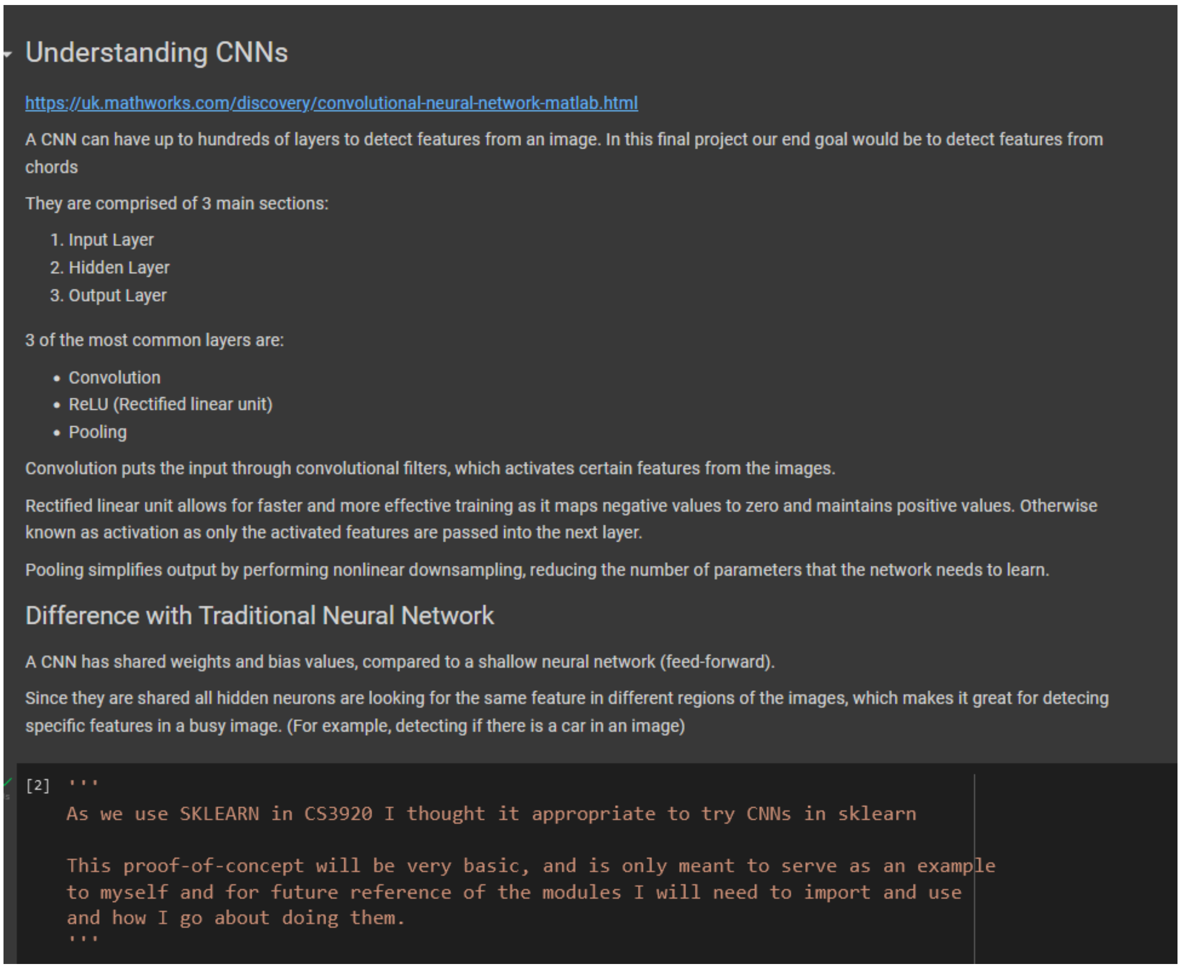


Figure 5.25 from [Müller, FMP, Springer 2015]

The above figure shows a hidden Markov model, state-dependent probabilities are shown by the dashed red lines. The initial states can be chosen by a musical expert. We can see above the HMM telling us the likelihood of a state returning a particular chroma feature.

2.3.8 Convolutional Neural Networks & Proof-of-Concept



Understanding CNNs

https://uk.mathworks.com/discovery/convolutional-neural-network-matlab.html

A CNN can have up to hundreds of layers to detect features from an image. In this final project our end goal would be to detect features from chords

They are comprised of 3 main sections:

1. Input Layer
2. Hidden Layer
3. Output Layer

3 of the most common layers are:

- Convolution
- ReLU (Rectified linear unit)
- Pooling

Convolution puts the input through convolutional filters, which activates certain features from the images.

Rectified linear unit allows for faster and more effective training as it maps negative values to zero and maintains positive values. Otherwise known as activation as only the activated features are passed into the next layer.

Pooling simplifies output by performing nonlinear downsampling, reducing the number of parameters that the network needs to learn.

Difference with Traditional Neural Network

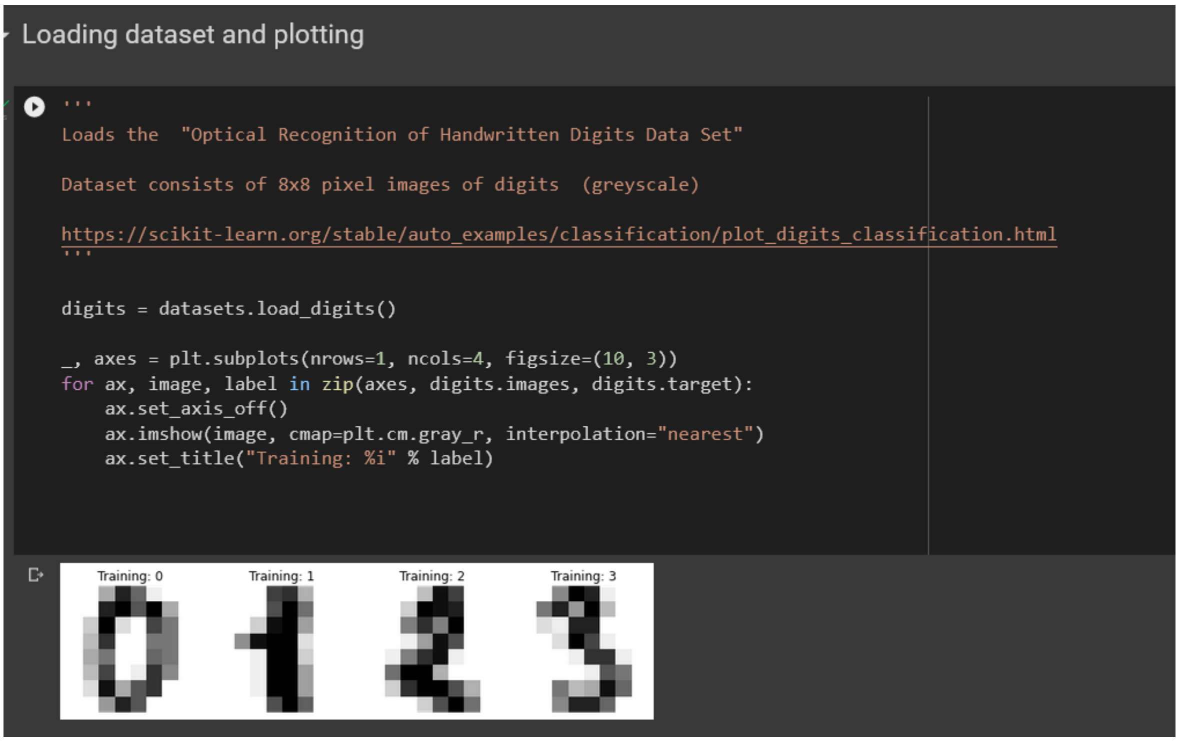
A CNN has shared weights and bias values, compared to a shallow neural network (feed-forward).

Since they are shared all hidden neurons are looking for the same feature in different regions of the images, which makes it great for detecing specific features in a busy image. (For example, detecting if there is a car in an image)

```
[2] '''
    As we use SKLEARN in CS3920 I thought it appropriate to try CNNs in sklearn

    This proof-of-concept will be very basic, and is only meant to serve as an example
    to myself and for future reference of the modules I will need to import and use
    and how I go about doing them.
    '''
```

As convolutional neural networks work so well for images, it is no wonder that chroma features (which are often expressed as images) would work well for chord recognition.

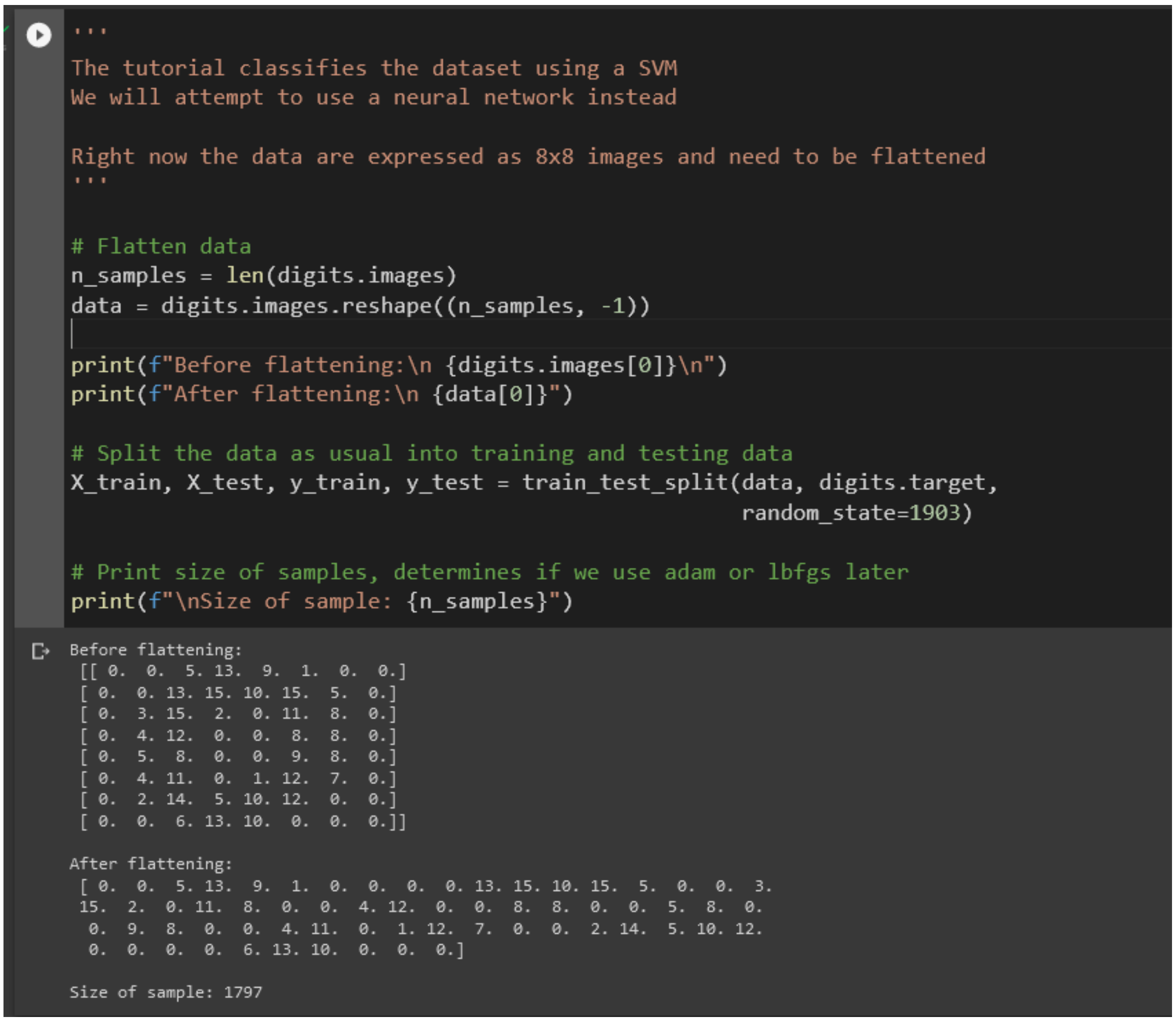
In my short proof-of-concept I wanted to attempt to learn how sklearn handles ConvNets as we use the library in CS3920 and gives me practice for what I shall expect.



Loading dataset and plotting

```
'''
Loads the  "Optical Recognition of Handwritten Digits Data Set"

Dataset consists of 8x8 pixel images of digits  (greyscale)

https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html
'''

digits = datasets.load_digits()

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("Training: %i" % label)
```

I used the “Optical Recogniton of Handwritten Digits Data Set” as it is a simple dataset that does require much pre-processing and eliminates the need to obtain chroma features as I have yet to find a suitable dataset.

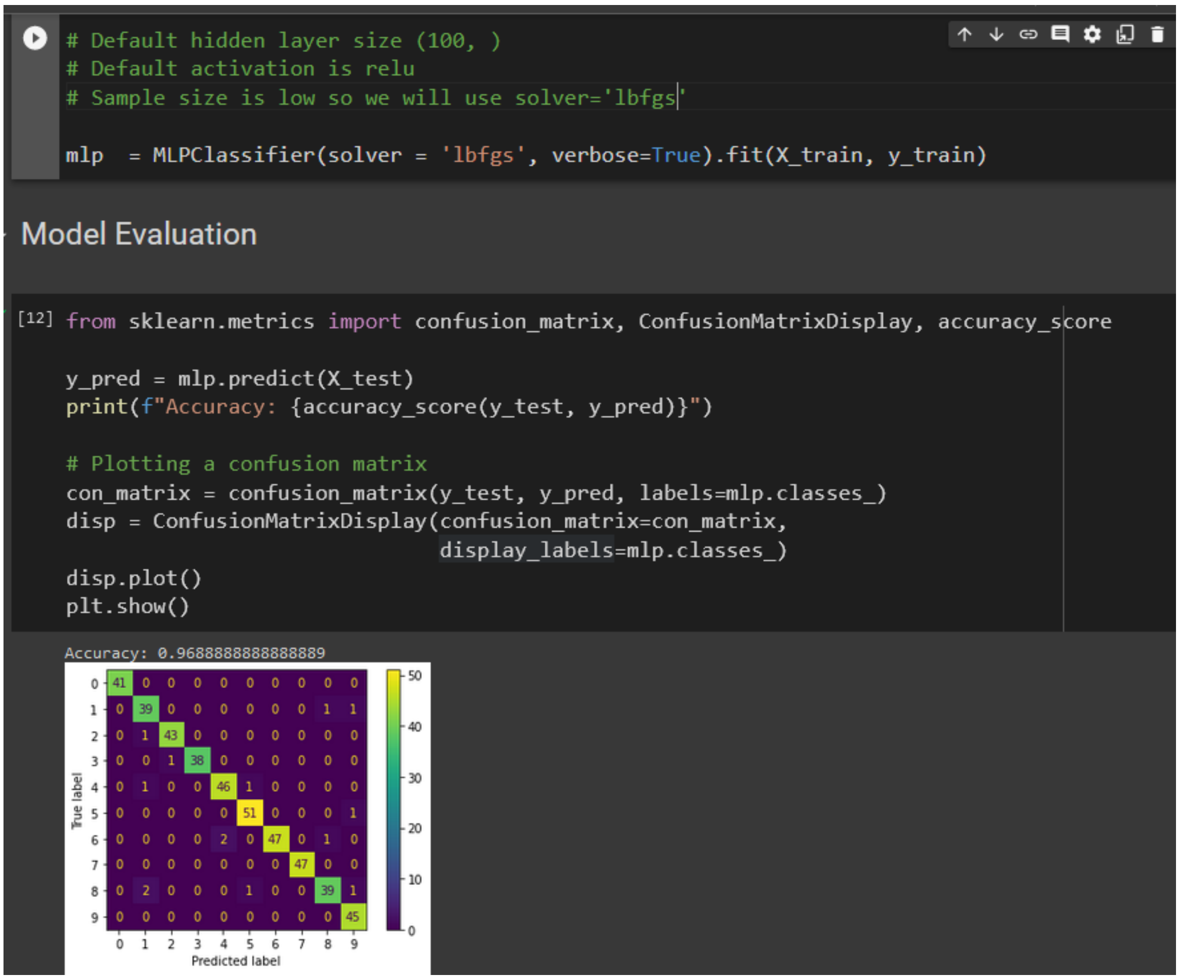


```
'''
The tutorial classifies the dataset using a SVM
We will attempt to use a neural network instead

Right now the data are expressed as 8x8 images and need to be flattened
'''

# Flatten data
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

print(f"Before flattening:\n {digits.images[0]}\n")
print(f"After flattening:\n {data[0]}")

# Split the data as usual into training and testing data
X_train, X_test, y_train, y_test = train_test_split(data, digits.target,
                                                      random_state=1903)

# Print size of samples, determines if we use adam or lbfgs later
print(f"\nSize of sample: {n_samples}")
```

```
Before flattening:
 [[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]

After flattening:
 [ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3.
 15.  2.  0. 11.  8.  0.  0.  4. 12.  0.  0.  8.  8.  0.  0.  5.  8.  0.
  0.  9.  8.  0.  0.  4. 11.  0.  1. 12.  7.  0.  0.  2. 14.  5. 10. 12.
  0.  0.  0.  0.  6. 13. 10.  0.  0.  0.]

Size of sample: 1797
```

At this stage the data is represented by an 8x8 array which will not work as in input to the CNN therfore we must flatten our input so that it can pass into the first later as one long feature vector that the “neurons” can process.

Once that is done, we split the data 25/75(test/train) which is the default parameters for sklearn, and set the random state to my birthday (DD/MM). Additonally, I print the size of the samples to determine if the solver for weight optimisation should be “adam” or “lbfgs”. Adam works well on larger datasets where lbfgs performs better for smaller datasets. The digits dataset contains 1797 images, so it is a relativetively small dataset so “lbfgs” is used.



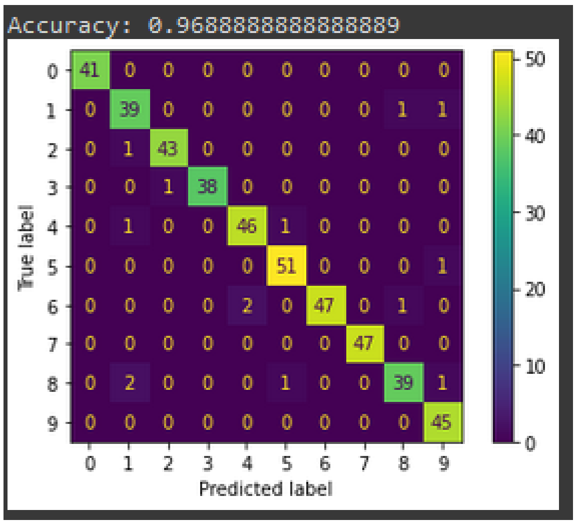
```
# Default hidden layer size (100, )
# Default activation is relu
# Sample size is low so we will use solver='lbfgs'

mlp  = MLPClassifier(solver = 'lbfgs', verbose=True).fit(X_train, y_train)
```

Model Evaluation

```
[12] from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_score

     y_pred = mlp.predict(X_test)
     print(f"Accuracy: {accuracy_score(y_test, y_pred)}")

     # Plotting a confusion matrix
     con_matrix = confusion_matrix(y_test, y_pred, labels=mlp.classes_)
     disp = ConfusionMatrixDisplay(confusion_matrix=con_matrix,
                                   display_labels=mlp.classes_)

     disp.plot()
     plt.show()
```

```
Accuracy: 0.9688888888888889
```

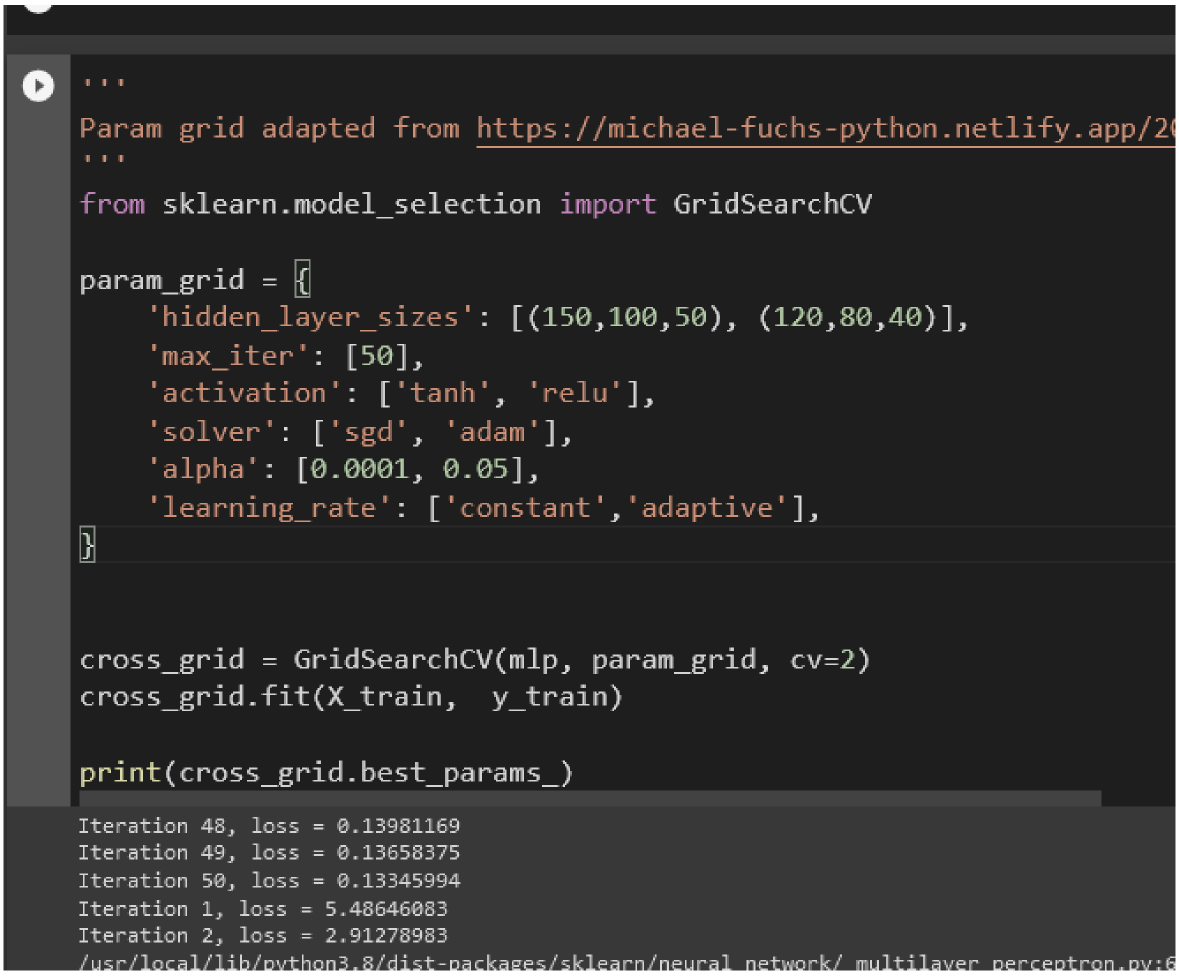
I then proceed to fit our training data on the Multi Layer Perceptron classifier – with parameters all set to default excluding the solver which for reasons stated above, I set to “lbfgs”. I then imported sklearns confuson matrix metric to observe the results.



```
Accuracy: 0.9688888888888889
```

True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	41	0	0	0	0	0	0	0	0	0
1	0	39	0	0	0	0	0	0	1	1
2	0	1	43	0	0	0	0	0	0	0
3	0	0	1	38	0	0	0	0	0	0
4	0	1	0	0	46	1	0	0	0	0
5	0	0	0	0	0	51	0	0	0	1
6	0	0	0	0	2	0	47	0	1	0
7	0	0	0	0	0	0	0	47	0	0
8	0	2	0	0	0	1	0	0	39	1
9	0	0	0	0	0	0	0	0	0	45

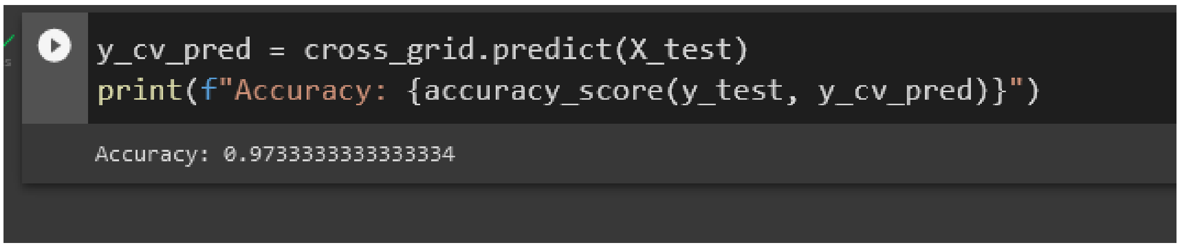
The accuracy of the classifer is very high and the confusion matrix highlights this with the class with the highest false negative/positives being the class 1 causing the most confusion.



```
'''
Param grid adapted from https://michael-fuchs-python.netlify.app/2
'''
from sklearn.model_selection import GridSearchCV

param_grid = {
    'hidden_layer_sizes': [(150,100,50), (120,80,40)],
    'max_iter': [50],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}


cross_grid = GridSearchCV(mlp, param_grid, cv=2)
cross_grid.fit(X_train,  y_train)

print(cross_grid.best_params_)
```

```
Iteration 48, loss = 0.13981169
Iteration 49, loss = 0.13658375
Iteration 50, loss = 0.13345994
Iteration 1, loss = 5.48646083
Iteration 2, loss = 2.91278983
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:6
```

I then decided to do cross validaton testing to see what the impact would be on the accuracy. I estimated only an incremental increase as I only run cross validation twice and with little variation on parameters, however this was in the interest of time and performance.



```
y_cv_pred = cross_grid.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_cv_pred)}")
```

```
Accuracy: 0.9733333333333334
```

The accuracy increased by 0.005 percent which is a very minimal increase.

This approach shows promise however with using chroma features as identifying digits and chords, while being very different problems have the same underlying solution. My main concern that quite a lot of previous works expressed is the lack of labelled data, pre-processed data and data without noise. I do not believe I will be able to achieve near 90 percent accuraccy, if I use previous works figures it should be possible to achieve approximately 60 percent or more.

2.3.9 Bibliography and Citations

[1] B. McFee et al., “librosa: Audio and Music Signal Analysis in Python,” presented at the Python in Science Conference, Austin, Texas, 2015, pp. 18–24. doi: 10.25080/Majora-7b98e3ed-003.

[2] M. Müller, “Chord Recognition,” in Fundamentals of Music Processing: Using Python and Jupyter Notebooks, M. Müller, Ed. Cham: Springer International Publishing, 2021, pp. 241–308. doi: 10.1007/978-3-030-69808-9_5.

[3] E. J. Humphrey and J. P. Bello, “Rethinking Automatic Chord Recognition with Convolutional Neural Networks,” in 2012 11th International Conference on Machine Learning and Applications, Dec. 2012, vol. 2, pp. 357–362. doi: 10.1109/ICMLA.2012.220.

[4] J. P. Bello and J. Pickens, “A Robust Mid-level Representation for Harmonic Content in Music Signals,” p. 8.

[5] T. Cho, R. J. Weiss, and J. P. Bello, “EXPLORING COMMON VARIATIONS IN STATE OF THE ART CHORD RECOGNITION SYSTEMS,” p. 8.

[6] “Harmonic pitch class profiles,” Wikipedia. Aug. 12, 2022. Accessed: Dec. 06, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Harmonic_pitch_class_profiles&oldid=1104073643

[7] “IBM Documentation,” Oct. 13, 2021. https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/mapms/1_cloud?topic=detectors-window-size-scheduling

[8] “Windowing - Introduction to Speech Processing - Aalto University Wiki.” https://wiki.aalto.fi/display/ITSP/Windowing (accessed Dec. 06, 2022).

[9] “How window size determines frequency resolution,” Merlijn van Veen. https://www.merlijnvanveen.nl/en/study-hall/21-how-window-size-determines-frequency-resolution

[10] “How a guitar works.” https://newt.phys.unsw.edu.au/music/guitar/guitarintro.html

[1] “Transfer Learning with Deep Network Designer - MATLAB & Simulink - MathWorks United Kingdom.” https://uk.mathworks.com/help/deeplearning/ug/transfer-learning-with-deep-network-designer.html (accessed Mar. 28, 2023).

[2] “Markov Chains | Brilliant Math & Science Wiki.” https://brilliant.org/wiki/markov-chains/ (accessed Dec. 06, 2022).

[3] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.” arXiv, Apr. 16, 2017. doi: 10.48550/arXiv.1704.04861.

[4] S. Hegde, “An Introduction to Separable Convolutions,” *Analytics Vidhya*, Nov. 24, 2021. https://www.analyticsvidhya.com/blog/2021/11/an-introduction-to-separable-convolutions/ (accessed Mar. 29, 2023).

[5] C.-F. Wang, “A Newbie’s Introduction to Convolutional Neural Networks,” *Medium*, Jul. 28, 2018. https://towardsdatascience.com/what-is-a-neural-network-6010edabde2b (accessed Mar. 29, 2023).

[6] “A Guide to AlexNet, VGG16, and GoogleNet,” *Paperspace Blog*, Jun. 01, 2020. https://blog.paperspace.com/popular-deep-learning-architectures-alexnet-vgg-googlenet/ (accessed Mar. 29, 2023).

[7] C. Szegedy *et al.*, “Going Deeper with Convolutions.” arXiv, Sep. 16, 2014. doi: 10.48550/arXiv.1409.4842.

[8] “ML | Inception Network V1,” *GeeksforGeeks*, Jan. 14, 2020. https://www.geeksforgeeks.org/ml-inception-network-v1/ (accessed Mar. 29, 2023).

[9] C.-F. Wang, “A Basic Introduction to Separable Convolutions,” *Medium*, Aug. 14, 2018. https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728 (accessed Mar. 29, 2023).

[10] J. Brownlee, “10 Clustering Algorithms With Python,” *MachineLearningMastery.com*, Apr. 05, 2020. https://machinelearningmastery.com/clustering-algorithms-with-python/ (accessed Mar. 30, 2023).

[11] “What is Supervised Learning? | IBM.” https://www.ibm.com/topics/supervised-learning (accessed Mar. 30, 2023).

[12] “Semi-Supervised Learning for Classification - MATLAB & Simulink - MathWorks United Kingdom.” https://uk.mathworks.com/help/stats/semi-supervised-learning-for-classification.html (accessed Mar. 30, 2023).

[13] “Reinforcement learning,” *GeeksforGeeks*, Apr. 25, 2018. https://www.geeksforgeeks.org/what-is-reinforcement-learning/ (accessed Mar. 30, 2023).

[14] “Can we do without labeled data? (Un)supervised ML,” *dida Machine Learning*. https://dida.do/blog/can-we-do-without-labeled-data-un-supervised-ml (accessed Mar. 30, 2023).

[15] M. Stewart, “Five Big Problems With Labeled Data,” *Zumo Labs*, Feb. 11, 2021. https://www.zumolabs.ai/post/five-big-problems-with-labeled-data (accessed Mar. 30, 2023).

[16] O. Alonso, “Challenges with Label Quality for Supervised Learning,” *J. Data Inf. Qual.*, vol. 6, no. 1, p. 2:1-2:3, Mar. 2015, doi: 10.1145/2724721.

[17] “IDMT-SMT-Chords - Fraunhofer IDMT,” *Fraunhofer Institute for Digital Media Technology IDMT*. https://www.idmt.fraunhofer.de/en/publications/datasets/chords.html (accessed Mar. 29, 2023).

[18] “GuitarSet,” *GuitarSet*. https://guitarset.weebly.com/ (accessed Mar. 29, 2023).

[19] M. C. Monard, “Learning with Skewed Class Distributions”.

[20] C. X. Ling and V. S. Sheng, “Cost-Sensitive Learning,” in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds., Boston, MA: Springer US, 2010, pp. 231–235. doi: 10.1007/978-0-387-30164-8_181.

[21] "2. Cost-sensitive learning — Reproducible Machine Learning for Credit Card Fraud detection - Practical handbook." https://fraud-detection-handbook.github.io/fraud-detection-handbook/Chapter_6_ImbalancedLearning/CostSensitive.html (accessed Mar. 30, 2023).

[22] M. Koziarski, "CSMOUTE: Combined Synthetic Oversampling and Undersampling Technique for Imbalanced Data Classification." arXiv, Apr. 17, 2021. Accessed: Mar. 30, 2023. [Online]. Available: http://arxiv.org/abs/2004.03409

[23] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002, doi: 10.1613/jair.953.

[24] K. Choi, G. Fazekas, M. Sandler, and K. Cho, "Transfer learning for music classification and regression tasks." arXiv, Sep. 13, 2017. doi: 10.48550/arXiv.1703.09179.

[25] L. Tran, S. Zhang, and E. Zhou, "CNN Transfer Learning for Visual Guitar Chord Classification".

[26] I. Junejo and N. Ahmed, "Depthwise Separable Convolutional Neural Networks for Pedestrian Attribute Recognition," *SN Comput. Sci.*, vol. 2, Apr. 2021, doi: 10.1007/s42979-021-00493-z.

2.4 Early Deliverable 2 - Analysis of existing literature surrounding chord recognition and implementations

2.4.1 Aims

Similarly, to deliverable 1, there exists plentiful machine and deep learning algorithms alongside different sound representations of which I will use only 1 in my final deliverable. To that end, the aim of deliverable 2 was to develop an understanding of how automatic chord recognition algorithms are built and the best practices. It will involve research of existing literature, market research and code examples. For ease, I will be using Python to test and develop my program, as such, I will be applying these algorithms at a high level abstracted using libraries. I aim to at the end of this report be able to implement a proof-of-concept python program that applies the techniques applied in literature to sample MNIST datasets.

2.4.2 Objectives

1. To compare different machine learning and deep learning algorithms implementations in Python on different representations of audio

2. To compare different audio processing techniques for best accuracy and speed for automatic chord recognition

3. To investigate and evaluate the feasibility of selected machine learning techniques using a small proof-of-concept and image recognition tasks (to be discussed in deliverable 3 and proof-of-concept)

4. To assess ability to create a real time automatic chord recognition system based on complexity of machine learning approach (from chord recognition standpoint)

2.4.3 Abstract

In this report, I offer a broad analysis of existing literature surrounding chord recognition and implementations. In my research I focused on these processes: Data preparation, algorithm selections, how to train data, what results I can expect.

This deliverable will not focus on the machine learning algorithms used but rather the ideas behind their use alongside their sound processing techniques. Additionally, I briefly explore methods for predicting chord progressions using natural language processing techniques. Finally, I evaluate the effectiveness of the implementations I have read and offer the path that I will take.

Keywords: Convolutional Neural Networks, Hidden Markov Models, Multitask Learning, Transfer Learning, Supervised Learning, Machine learning, Deep learning, Support Vector Machines, Fourier Transform, Constant Q-Transform, Harmonic Pitch Class Profile, Mel-Frequency Cepstral Coefficients, Mel-Spectrogram, Natural Language Processing, N-gram, Chroma, Chroma Features,

2.4.4 Summary of Research

In the field of music information retrieval and automatic chord recognition, there exist multiple implementations of machine learning algorithms. Currently, a large amount of existing literature has used a mixture of CNN configurations with spectral coefficients such as Constant Q-Transform (CQT) [1] and the Mel-frequency cepstral coefficients (MFCCs) [2]. Chroma features are then extracted to be used in the learning and recognition stages. In python, a library called librosa is often used for this purpose [3]

A quote that encapsulates the same general 3 ideas that all previous works have aimed to follow is:

“Fast frame-level features are calculated over short-time observations of an audio signal, instantaneous feature vectors are assigned to chord classes, and post-filtering is performed to identify the best chord classification path.” [4]

While each work attempts a slightly different approach, the difference arises from either the pre-processing of the chord data, or the audio representation form chosen for the problem. This highlights the need for better chroma features to be fed into the algorithms, an augmented dataset or unsupervised learning? I look to have this question answered in my final project report.

Below I have briefly detailed a selection of interesting approaches to the issue, along with an explanation and what audio representation they use, interesting points will be highlighted in later sections.

- ConvNets using transfer learning via multi-layer representations using a source task – music tagging, due to incredible depth of labelled data [5]
    - Activations of final hidden layer in conjunction with all intermediate layers are used

    - Mel-spectrograms are the chosen audio representation because "it is psychologically relevant and computationally efficient"

    - MFCC is used, as it is adopted in many works of music information retrieval

    - Humphrey & Bello [4] use CNNs in an unconventional way by training the network on 5 seconds of pitch spectra to eliminate data wastage from post-filtering of the audio. This approach allows for all data to be used in the network than making assumptions about what might be deemed useful. It is worth noting that for this approach they used the CQT to produce their pitch spectra

- Hidden Markov Models (HMMs) and N-gram models due to better similarity with music theory [6]

    - This approach is supported by Juan P. Bello & Jeremy Pickens observations [7] whereby paying close mind to music theory, they first generate a chromagram using the QCT [8], initialise their state transition matrix of their HMM with musical knowledge of likely chord transitions. It should be noted that Bello and Pickens opted for unsupervised training due to lack of labelled training data (which may be an area of consideration as the lack of labelled data is often an issue for automatic chord recognition)

    - Pitch class profiles vectors (PCPs) [9] are used by Sheh & Ellis [10] in their paper outlining the use of HMMs trained with an expectation algorithm of which chord labels were their only input data and via the Baum-Welch algorithm.

    - Lee & Slaney [11] use a 36-state HMM, where each state represents a single chord. Features are assumed to be unrelated and so they use diagonal covariance matrix. Since, state transitions obey the first-order Markov property, they opted to apply the Viterbi algorithm to find the optimal path e.g., maximum likelihood of a chord transition.

        - They overcome the issue Bello & Pickens encountered by using a sample-based synthesiser to create midi files as well as handcrafted labels, allowing them to use a larger set of labelled data. It is worth noting that variations of a chord such as dominant seventh chords were recognised as the original triad (C7 -> C) and they treated this as a correct classification.

        - This may not suffice for my system as we want the recognition to be as accurate as possible.

- SVMs were found to be extremely effective (alongside feed forward neural networks) when attempting to classify instruments based on their timbre [12]. Like the above, timbre was represented by chroma features

A large majority of the papers I have read utilise CNNs or HMMs. CNNs are often associated with transfer learning combined with QCT which shows great promise to the "chord recognition" section of my project, however I am also concerned with predicting chord sequences. For this, HMMs appear to work quite well and in their design intuitively make sense to use as they represent uncertainty between chords but can create links using musical knowledge in the state transition model. Furthermore, regardless of which HMM method we select the optimal path of chord transitions are found using the Viterbi algorithm.[13]

Additionally, Cheng et al. [6] describes a method of optimal chord progression selection via a N-gram model which I found quite interesting as they claim the time complexity of their method is one order less than the Viterbi algorithm. This requires further exploration however as language modelling is not an area in which I am familiar.

As such, I will attempt to adopt the HMM approach, should this prove to be too difficult considering my lack of experience I will instead pursue CNNs via transfer learning as I have developed a rudimentary implementation prior to commencing this project.

2.4.5 Bibliography and Citations

[1] K. O'Hanlon and M. B. Sandler, "Comparing CQT and Reassignment Based Chroma Features for Template-based Automatic Chord Recognition," in ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), May 2019, pp. 860–864. doi: 10.1109/ICASSP.2019.8682774.

[2] T. Cho and J. P. Bello, "A FEATURE SMOOTHING METHOD FOR CHORD RECOGNITION USING RECURRENCE PLOTS," Oral Sess., p. 6, 2011.

[3] B. McFee et al., "librosa: Audio and Music Signal Analysis in Python," presented at the Python in Science Conference, Austin, Texas, 2015, pp. 18–24. doi: 10.25080/Majora-7b98e3ed-003.

[4] E. J. Humphrey and J. P. Bello, "Rethinking Automatic Chord Recognition with Convolutional Neural Networks," in 2012 11th International Conference on Machine Learning and Applications, Dec. 2012, vol. 2, pp. 357–362. doi: 10.1109/ICMLA.2012.220.

[5] K. Choi, G. Fazekas, M. Sandler, and K. Cho, "Transfer learning for music classification and regression tasks." arXiv, Sep. 13, 2017. doi: 10.48550/arXiv.1703.09179.

[6] H.-T. Cheng, Y.-H. Yang, Y.-C. Lin, I.-B. Liao, and H. H. Chen, "Automatic chord recognition for music classification and retrieval," in 2008 IEEE International Conference on Multimedia and Expo, Jun. 2008, pp. 1505–1508. doi: 10.1109/ICME.2008.4607732.

[7] J. P. Bello and J. Pickens, "A Robust Mid-level Representation for Harmonic Content in Music Signals," p. 8.

[8] J. C. Brown, "Calculation of a constant Q spectral transform," J. Acoust. Soc. Am., vol. 89, no. 1, pp. 425–434, Jan. 1991, doi: 10.1121/1.400476.

[9] T. Fujishima, "Realtime Chord Recognition of Musical Sound: a System Using Common Lisp Music," 1999. https://quod.lib.umich.edu/i/icmc/bbp2372.1999.446/--realtime-chord-recognition-of-musical-sound-a-system-using?view=image (accessed Oct. 06, 2022).

[10] A. Sheh and D. P. W. Ellis, "Chord Segmentation and Recognition using EM-Trained Hidden Markov Models," p. 7.

[11] K. Lee and M. Slaney, "Automatic Chord Recognition from Audio Using a HMM with Supervised Learning.," Jan. 2006, pp. 133–137.

[12] H. Ezzaidi, M. Bahoura, and G. Hall, "Towards a Characterization of Musical Timbre Based on Chroma Contours," presented at the Communications in Computer and Information Science, Dec. 2012, vol. 322, pp. 162–171. doi: 10.1007/978-3-642-35326-0_17.

[13] H. Papadopoulos and G. Peeters, "Large-Scale Study of Chord Estimation Algorithms Based on Chroma Representation and HMM," in 2007 International Workshop on Content-Based Multimedia Indexing, Talence, France, Jun. 2007, pp. 53–60. doi: 10.1109/CBMI.2007.385392.

[14] G. Gwardys and D. M. Grzywczak, "Deep Image Features in Music Information Retrieval," Int. J. Electron. Telecommun., vol. 60, no. 4, Art. no. 4, Dec. 2014.

[15] tyker1, "What is the difference between Constant-Q Transform and Wavelet Transform and which is better," Signal Processing Stack Exchange, Jun. 01, 2021. https://dsp.stackexchange.com/q/43811 (accessed Dec. 01, 2022).

[16] L. Camp, "Answer to 'What is the difference between Constant-Q Transform and Wavelet Transform and which is better,'" Signal Processing Stack Exchange, Jan. 24, 2018. https://dsp.stackexchange.com/a/46657 (accessed Dec. 01, 2022).

[17] C. Schörkhuber and A. Klapuri, "CONSTANT-Q TRANSFORM TOOLBOX FOR MUSIC PROCESSING," p. 8.

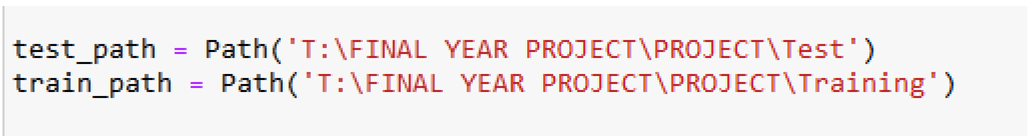
[18] N. Jiang, P. Grosche, V. Konz, and M. Muller, "Analyzing Chroma Feature Types for Automated Chord Recognition," p. 10, 2011.

[19] "Investigating style evolution of Western classical music: A computational approach." https://journals.sagepub.com/doi/epub/10.1177/1029864918757595 (accessed Dec. 06, 2022).

Chapter 3: **Setup + Instructions**

Prerequisites:

1. Python 3+
2. Any packages that are listed within any python notebook files must be installed.
    a. I will provide a requirements txt file that can be run with pip.
3. If using any file called Kaggle the dataset will need to be downloaded:
    a. https://www.kaggle.com/datasets/fabianavinci/guitar-chords-v3
    b. You'll notice that there are two lines of code:



```
test_path = Path('T:\FINAL YEAR PROJECT\PROJECT\Test')
train_path = Path('T:\FINAL YEAR PROJECT\PROJECT\Training')
```

    Replace these two lines with the location of your test and training folder after downloading the dataset.

4. If using any file called IDMT the dataset will need to be downloaded:
    a. https://www.idmt.fraunhofer.de/en/publications/datasets/chords.html
    b. The dataset was split into multiple wav files and saved on my local device. So, it may not be possible to obtain the same results. However, all outputs have been saved and the demo videos demonstrate the code.
5. If using any file called ESC50 the dataset will be downloaded to your system and all code should run provided you have installed all necessary imports.

Chapter 4: **Code & Models**

4.1 Introduction, Layout and Datasets

We will investigate 3 models: YAMNet (built off MobileNet_V1 architecture), GoogLeNet and VVGISH(VGG16).

All code will be developed and deployed on Jupyter Notebook and I have used anaconda to install the data science libraries I use as well as Python and Jupyter Notebook itself. For more information on Anaconda please visit: https://anaconda.org/

TensorFlow with Keras is my chosen machine learning framework due to ease, abundance of tutorials and compatibility with Jupyter. For more information on Tensorflow please visit: https://www.tensorflow.org/ . For more information on Keras please visit: https://keras.io/ . They are both well known and popular frameworks and often during my project I would refer to their user manuals for understanding machine learning concepts.

Throughout my project I used 3 datasets: ESC-50[3], a dataset that consists of 8 chords from Kaggle[4] and IDMT-SMT-Chords[5]**.** In short, ESC-50 is a labelled collection of environmental audio recordings, Kaggle dataset is a collection of 8 chords that have already been split into a training and testing folder and IDMT-SMT-CHORDS is a dataset that consist of 83 different chord classes however it does not have a large amount of data for each chord class.

I use the ESC-10 as a control variable to test the models to obtain a generalised view of accuracy and how to interact with the models. TensorFlow has an implantation of the ESC-10 available to download and through following their tutorial for YAMNet I was able to replicate training and testing splits for all models.

4.2 Project Run-Through

Please find the run-through of my project in this YouTube playlist: [https://www.youtube.com/playlist?list=PLFJMdLtTCP75AiwioLN_71uZT7rQYitK6]

Please note that file paths and some code may be different in the final submission. This is because I will need to amend my code to run on other machines apart from my own, for example, removing TensorFlow GPU and using CPU instead.

[3] https://github.com/karolpiczak/ESC-50

[4] https://www.kaggle.com/datasets/fabianavinci/guitar-chords-v3

[5] https://www.idmt.fraunhofer.de/en/publications/datasets/chords.html

4.3 YAMNet

4.3.1 YAMNet Theory

YAMNet is audio event classifier that takes audio waveforms as input and makes a prediction for each of the 521 audio events in the AudioSet ontology.[6] AudioSet is a large-scale data dataset of manually annotated audio events, created using 10 second YouTube Clips. It is built on the MobileNetV1 depth-wise-separable convolution architecture [3]. From my understanding, a depth-wise-separable convolution architecture is one that applies first depth-wise convolution and then pointwise convolution. Using the example of a 10x10xN image with no padding, if we apply a 3x3 convolution with a stride of 1 we end up with a 7x7 image. Next, we apply this 3x3x1 kernel and iterate through 1 channel (multiple 3x3x1 kernels) giving us a 7x7x1 image that can be stacked against all channels to create 7x7xN image. Now we apply pointwise convolution whereby a 1x1xN kernel is used that has a depth equal to our input image channel, for example if our depth is 1024 (which is the output length YAMNet uses) then our new output would be 7x7x1024. This is equivalent to a convolution with 1024 filters of 7x7 kernel shape, or 1024 1x1xN kernels that each separately produce an 7x7x1 image to obtain a shape of 7x7x1024. [3]–[5] Please see Figure 3 in the appendix for a visual reference.

Please see Figure 4 in the appendix for the full model description which details the input and outputs of YAMNet.

A brief description of YAMNet is as follows:

Inputs:

1-D Tensor or NumPy array – contains mono 16khz samples batches of waveform frames

Outputs:

Model returns a 3-tuple (scores, embeddings, log Mel spectrogram)

In Figure 4 of the appendix, you will find explanations for scores and the log Mel spectrogram, for the purpose of my project we are only interested in the embeddings. "Embeddings is a float32 Tensor of shape (N, 1024) containing per-frame embeddings, where the embedding vector is the average-pooled output that feeds into the final classifier layer." [3.] We can use the embedding vector for a shallow model, this uses YAMNet as a semantic feature extractor.

4.3.2 YAMNet Code

For the majority of YAMNet code experimentation I followed and amended the following tutorial: https://www.tensorflow.org/tutorials/audio/transfer_learning_audio

I made the two following changes:

1. The tutorial uses dogs and cats, I instead opted for thunderstorm and vacuum_cleaner

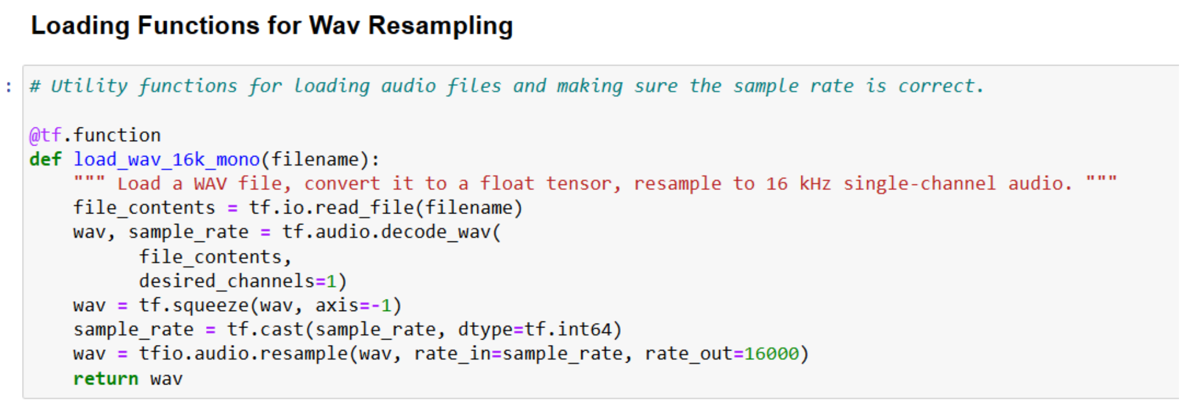
[6] https://tfhub.dev/google/yamnet/1

2. The tutorial only uses binary classification, since my chord datasets will be multi-categorical classification, I produced another example where we classify dogs, cats, yell, and children screaming.

```
# Load YamNet model
yamnet_model_handle = 'https://tfhub.dev/google/yamnet/1'
yamnet_model = hub.load(yamnet_model_handle)
```

The code to load in the model is straightforward, we simply download it from TensorFlows servers and load it into a variable.

As mentioned before YAMNet requies input to be a 1-D Tensor or NumPy array where samples are mono, sampled at a rate of 16Khz and normalised to values in the [-1.0, 1.0] range.



Loading Functions for Wav Resampling

```
# Utility functions for loading audio files and making sure the sample rate is correct.

@tf.function
def load_wav_16k_mono(filename):
    """ Load a WAV file, convert it to a float tensor, resample to 16 kHz single-channel audio. """
    file_contents = tf.io.read_file(filename)
    wav, sample_rate = tf.audio.decode_wav(
          file_contents,
          desired_channels=1)
    wav = tf.squeeze(wav, axis=-1)
    sample_rate = tf.cast(sample_rate, dtype=tf.int64)
    wav = tfio.audio.resample(wav, rate_in=sample_rate, rate_out=16000)
    return wav
```

This returns a normalised wav file with the constraints mentioned above.

First, I experimented with the control dataset (ESC-50) and followed the steps below:

I have written variations that I applied for the Kaggle and

1. Downloaded the dataset from TensorFlow and extracted it locally.

2. Using pandas, created a DataFrame object using the csv located in the ESC-50 folder once downloaded.

    a. I then explored the dataset taking note of how the DataFrame is structured as this would serve to be my basis for my own dataset preparation.

3. Filtered the data and apply 3 main transformations:

    a. Filter out rows and only take my selected classes.

    b. Amend the filename to have the full path, which will make loading induvial wav files much easier.

    c. Change the targets to be within a specific range, since I am only selecting a subcategory of classes.

4. Use the load_wav_16k_mono and prepare the wav data for the model.

   a. Embeddings extracted from the wav data will be in the form (N, 1024) where N is the number of frames that YAMNet found [1 for each 0.48 seconds of audio]

   b. The model will use each 1 frame as input, therefore within the Tensor we must map the same label with the same frames that correspond to the label. In the case of the ESC-50 database this entails also making sure the folds column remains the same so we do not mix frames into different folds.

      i. For the chord datasets there are no folds so that section is irrelevant, however, we still map the labels to the correct frames.

5. We then split the dataset into train, validation, and test sets. For the Kaggle dataset it is already split into train and test, I apply another split to the train data to create validation data so I can monitor for overfitting.

   a. We then apply batching to the datasets and shuffle the training dataset

      i. Batching is useful as it allows us to train in chunks, effectively passing the network a number of samples at a time at each epoch. This allows us to update the model's weights and parameters until we have reached the end. I used a batch of 32 for the ESC-50 database, 32 for the Kaggle dataset, and 64 for the IDMT dataset.

6. Next the model is created, it is a simple sequential model with one hidden layer and N outputs corresponding to the label size. It is important to note that the input size for YAMNet must be 1024.

7. The model is then tested against the test dataset and a graph is generated representing loss and accuracy across epochs to give a general overview of performance.

Please look at Project Run-Through for a demonstration.

4.4 GoogLeNet

4.4.1 GoogLeNet Theory

GoogLeNet is a CNN that is 22 layers deep [7] and is a network that has been trained on ImageNet. It has been trained to classify images into 1000 object categories, with model inputs being 224x224 images. It also was the winner of the ILSVRC 2014 image classification challenge.[8]

GoogLeNet was also the introduction of the inception network, Inception v1.

Inception networks were introduced to tackle the issue of kernel selection for information where the variation of the location of information changes dramatically. For example, if information in an

[7] https://uk.mathworks.com/help/deeplearning/ref/googlenet.html

[8] https://image-net.org/challenges/LSVRC/2014/results

image is spread globally then a larger kernel would be preferred, with the opposite holding true as well. [6]

The typical solution for CNNs to obtain better performance is to increase the size of the network, this can involve increasing both the depth and width of the network. This leads to two major issues: overfitting and computational expenses. Overfitting occurs due to the increased epochs and network complexity not being equal to the quality and quantity of labelled examples present in the dataset. Overlapping, passing gradient updates, and chaining through deep convolution layers is an expensive process with Szegedy et al stating “if two convolutional layers are chained, any uniform increase in the number of their filters results in a quadratic increase of computation”[7]

GoogLeNet aimed to solve this by using multiple kernel sizes on a single input of which is concatenated and passed onto the next layer, they also use a 1x1 convolution to reduce the number of channels before being passed into the 3x3 and 5x5 (the 1x1 convolution layer is added after max pooling layer).[8] This has the effect of the network growing wider as opposed to growing deeper, addressing computational expensiveness, and reducing the number of parameters. [9]

A diagram showing both the full network and inception modules can be found in the appendix: Figure 6 **Figure 7**

I will not implement GoogLeNet as I wish to work strictly on audio embeddings and representations instead of image recognition.

4.5 VGG16 – VGGISH

4.5.1 VGG16 – VGGISH Theory

VGG16 is a type of CNN that consists of 16 layers – hence VGG”16” - it is a variant of the VGG model. It has 3x3 convolutions and the network structure can be visualised here: http://ethereon.github.io/netscope/#/gist/dc5003de6943ea5a6b8b.

VGG standing for Visual Geometry Group is a set of models released by Oxford university as a project to investigate how ConvNet depth affects their accuracy in large-scale image recognition setting. [9] It was trained on a subset of ImageNet and like YAMNet can provide detect general features about its domain (images). In the 2014 ImageNet Classification Challenge VGG16 achieved a 92.7% classification on the ILSVRC-2012 dataset, the table of errors can be found at Figure 5.

[9] https://www.robots.ox.ac.uk/~vgg/research/very_deep/

Google (TensorFlow) offer a definition of a VGG-like model that was trained on AudioSet. This model was created to have more adapted weights for audio waveforms.[10]

4.5.2 VGG16 – VGGISH Code

The sequence of steps is the same for YAMNet with some minor differences:

1. As I am not following a tutorial for VVGISH I instead use a very similar process to YAMNet, however I use KerasTuner[11] which is a hyperparameter optimisation framework to help me create an initial shallow model.

```
# Load the model.
vgg_model = hub.load('https://tfhub.dev/google/vggish/1')

# Input: 3 seconds of silence as mono 16 kHz waveform samples.
waveform = np.zeros(3 * 16000, dtype=np.float32)

# Run the model, check the output.
embeddings = model(waveform)
embeddings.shape.assert_is_compatible_with([None, 128])
```

Loading the model is similar to YAMNet. However, a big difference is the model returns a 2-D float Tensor of shape (N, 128)[12].

Please look at Project Run-Through for a demonstration.

[10] https://github.com/tensorflow/models/tree/master/research/audioset/vggish

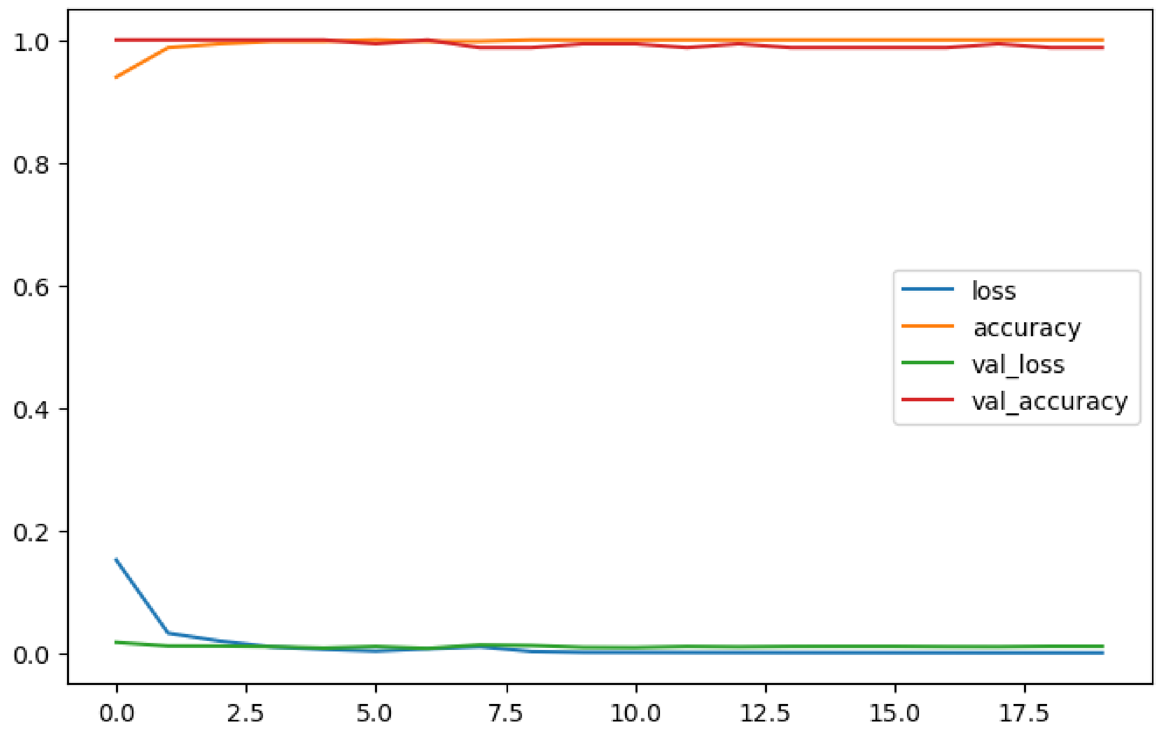
[11] https://keras.io/keras_tuner/

[12] https://tfhub.dev/google/vggish/1

Chapter 5: **Evaluation of Models**

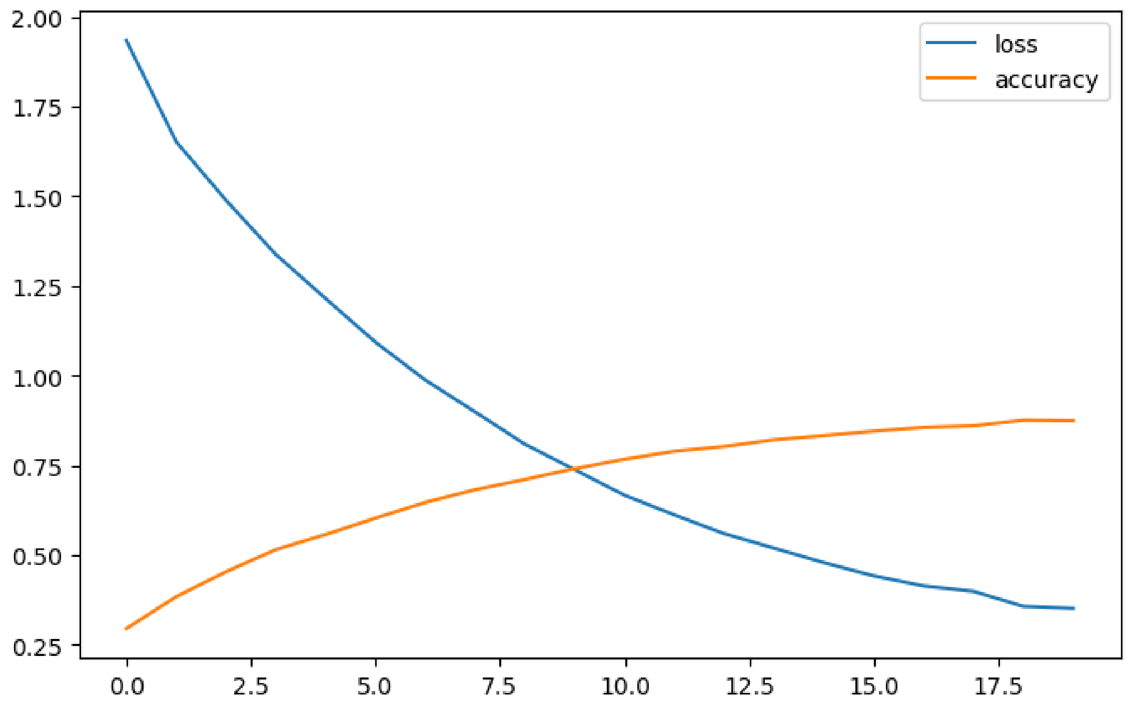
5.1 YAMNet

5.1.1 ESC-50



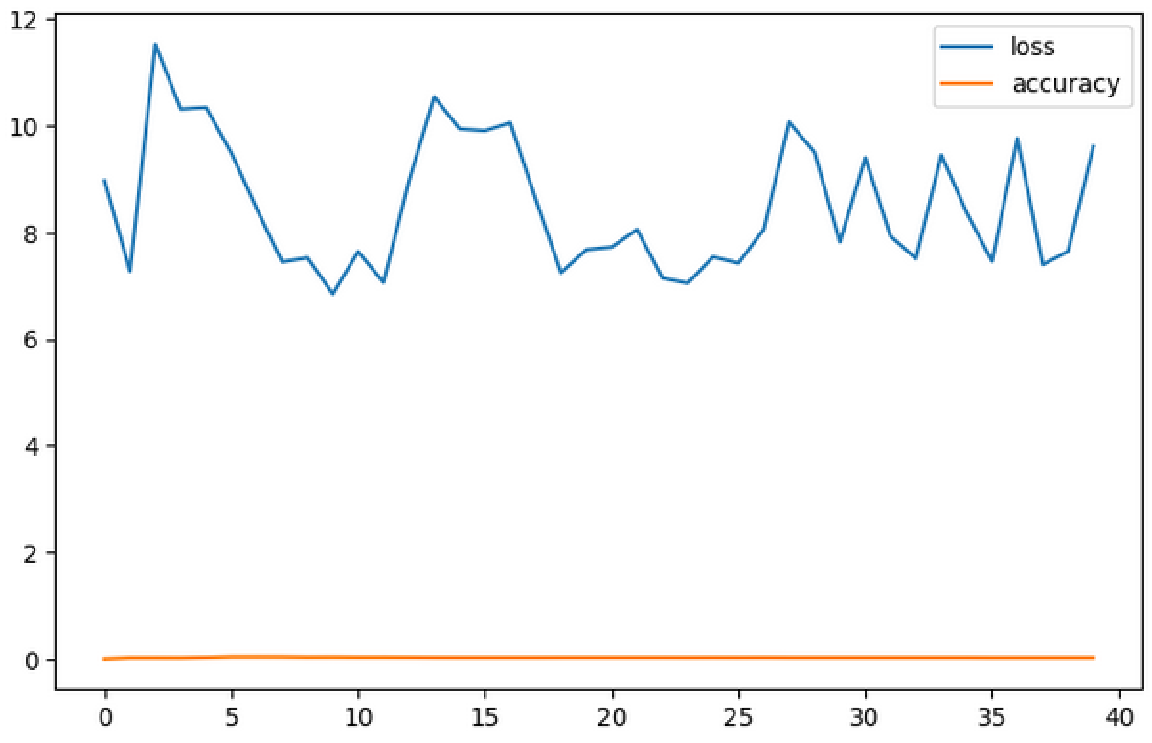
```
5/5 [==============================] - 0s 4ms/step - loss: 0.0438 - accuracy: 0.9750
Loss:  0.04377112537622452
Accuracy:  0.9750000238418579
```

5.1.2 Kaggle



```
190/190 [==============================] - 11s 56ms/step - loss: 3.1002 - accuracy: 0.4451
Loss:  3.100214719772339
Accuracy:  0.4451400339603424
```

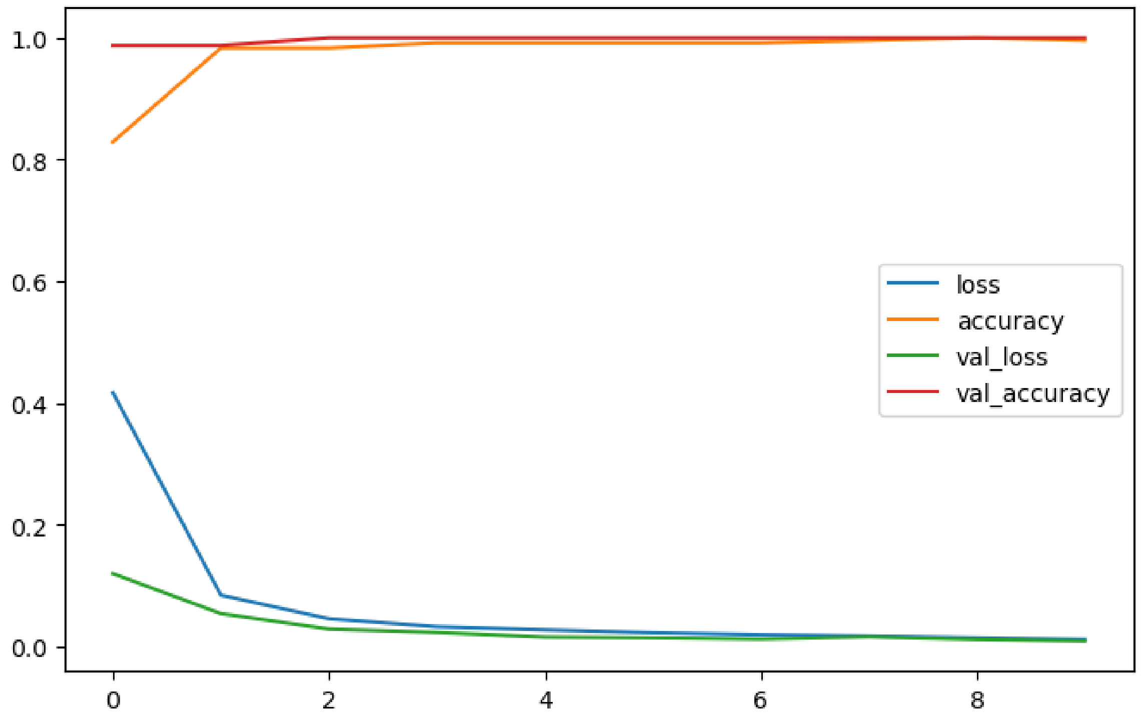
5.1.3 IDMT-CHORDS-SMT



```
2/2 [==============================] - 1s 35ms/step - loss: 9.7882 - accurac
y: 0.0000e+00
Loss:  9.78824520111084
Accuracy:  0.0
```

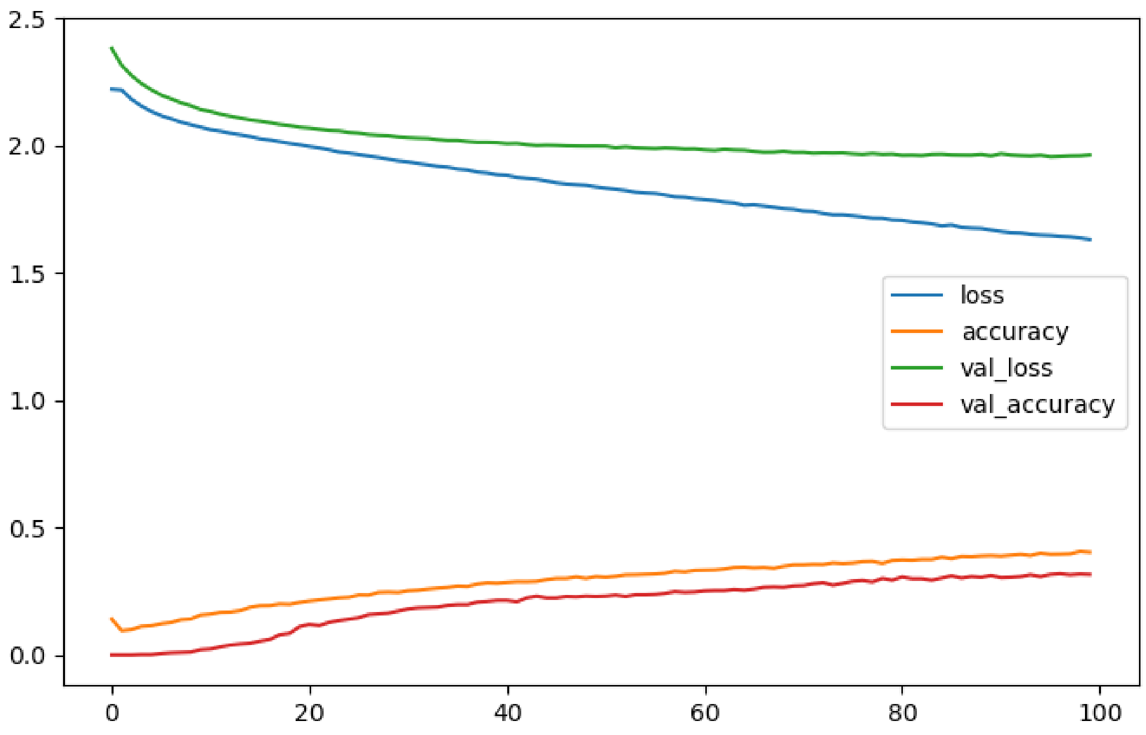
5.2 VGG16 - VGGISH

5.2.1 ESC-50



```
3/3 [==============================] - 0s 7ms/step - loss: 0.0251 - accuracy: 0.9875
Loss:  0.025112319737672806
Accuracy:  0.987500011920929
```

5.2.2 Kaggle



```
12/12 [==============================] - 0s 847us/step - loss: 1.6900 - accuracy: 0.3908
Loss:  1.6900261640548706
Accuracy:  0.39082059264183044
```

5.2.3 IDMT-CHORDS-SMT

```
1/1 [==============================] - 0s 16ms/step - loss: 9.1204 - accurac
y: 0.0000e+00
Loss:  9.120375633239746
Accuracy:  0.0
```

5.3 Summary

For both models we notice an extremely poor accuracy and testing accuracy on the IDMT dataset, average results on the Kaggle dataset and very good results on the ESC50 dataset. This is most likely directly correlated to the breadth and quality of data with IDMT having the worst in all categories with only approximately 3 chords per chord type. This would lead to overfitting or a model that has not learnt anything as it does not have the tools to learn anything which is what we see above. The Kaggle dataset has quite a lot of data for each of their chords so it could benefit from more rich data as none of these networks have been trained on guitar data before. Finally, we obtain very good results with ESC50 most likely due to the similarities between ImageNet (the data both models were originally trained on) and ESC50s dataset being an environmental sound database.

Overall, transfer learning shows promise towards recognising guitar chords, but can be significantly held back by not only quality of data but also quantity. I could rectify this by augmenting and creating data, but not only is this extremely tedious but also I do not have the resources to do so.

Chapter 6: **Professional Issues: Labelled Datasets, Overfitting and Accuracy**

Within the topic of Machine Learning there are primarily 4 types of learning problems: supervised, semi-supervised, unsupervised and reinforcement. Supervised learning is defined by its use of labelled datasets to train algorithms to make predictions and classify data. Unsupervised learning uses unlabelled datasets and instead works to find patterns to use for clustering or association problems. K-means and Gaussian Mixture Model are two popular clustering algorithms used for unsupervised learning. Semi-supervised is a combination of both methods and is used when labelled data is only a percentage of the overall dataset and relies on learning patterns with the help of supervised learning. Reinforcement learning is typically seen with agents whereby a model of reward when correct and consequence when wrong" is followed.[10]–[14]

My project focuses on supervised learning which is an industry standard when creating CNN models. However, supervised learning presents issues which are not easily solvable, these include:

1. Requires large amounts of labelled data.
    a. Labelled data must come from humans which can be prone to error, time consuming and costly.[15]
2. It is very prone to overfitting if your model is too complex or your input data is lacking in well structured data for your model to learn well.[16]
3. Computation time for supervised learning methods is high.

Within the domain of chord recognition, two chord datasets I found were [17], [18] which provide a good base dataset for starting audio classification, however, there are many pitfalls with both methods that lead to a poor model and accuracy when using transfer learning on a shallow model.

1. The amount of labelled data per chord class is quite small which leads to overfitting and poor testing accuracy.
2. The data has not been sorted beforehand which means that I must spend a substantial amount of time pre-processing data, which can and has led to errors that are not easily noticeable.
3. Data per chord class often has a poor distribution which entails the model having a more difficult time to predict certain chord classes and may skip the chord class altogether.
    a. This is known as a skewed class distribution.
    b. This will also cause a low error rate for the majority class and an incredibly high error rate for a minority class and as such, minimising the minority class is a priority as this increases the accuracy of the model.

Monrad et al [19] details 3 ways in which we can handle skewed class distribution:

1. Assign misclassification costs.
    a. In summary, the misclassification of the minority classes would lead to a higher cost than a misclassification of the majority classes. This is utilised in cost-sensitive learning systems.
        i. Cost-Sensitive Learning – takes the misclassification costs into consideration when training and therefore aim to minimise the total cost. It is used in multiple fields such as fraud detection, medicine and engineering.[20], [21]
        ii. In Keras, we usually test for validation error or training error. What we can do is add a cost for misclassification in the class_weight argument on the fit() function. [13]
2. Under-sampling.
    a. We can directly remove some samples of the majority class to artificially rebalance the class distributions.
3. Over-sampling.
    a. We can directly replicate some examples of the minority class to artificially increase the minority class distributions.

There are drawbacks to both under-sampling and over-sampling. Under-sampling may potentially throw away useful data points and over-sampling may lead to an increased possibility of overfitting. Some works aim to combine the two approaches whereby interpolation can occur between minority classes to create new classes that push the boundaries of the majority class space. Therefore by creating new samples in the minority space and removing some data from the majority class allowing us to overcome the overfitting of the model. Two popular techniques are SMOTE [Synthetic Minority Over-sampling Technique] and CSMOUTE [Combined Synthetic Oversampling and Undersampling Technique] [22], [23]

[13] https://machinelearningmastery.com/cost-sensitive-learning-for-imbalanced-classification/

Chapter 7: **Self-Assessment**

Managing expectations is a massive concept that had I planned for before my project would have led to much more consistent results, a better interim report and more time for exploration for different topics. I believe at the start of the project and within my initial deliverables I mentioned achieving an accuracy rate of 90+ percent which was incredibly idealistic. Upon further research I noticed that real world applications of these systems often achieve around 60-80 percent accuracy dependent on the task [24], [25].

Additionally, as evident in my frequent but largely gapped GitLab commits I had an issue of diving headfirst into the models and coding. This was a lapse in judgement caused by mismanaged expectations as I mentioned above, I had the impression that because it was "only CNNs" that it would be easy and prior knowledge of theory would not be needed. This assumption was very wrong and quite dangerous as it led me in the beginning to create code that was highly inefficient and would simply mimic the works the others without any personal touch or experimentation with certain variables such as model weights, epochs, altering how shallow or deep a network was and so forth (look at very early Gitlab commits). I learnt more Keras and TensorFlow features as the project went on, as well as learning more theory regarding CNNs and transfer learning that led me to improve accuracy readings from .012 to 0.46, although more could be done, I was happy at the result of my Keras journey.

Furthermore, I also assumed that guitar chord datasets would be readily available, labelled and sorted. This assumption was incorrect as there do not exist many large scale datasets for an exhaustive amount of chords, the two best datasets I found in my research were: [17] & [18]. However, as extensive as these datasets are, there was still the work of separating the data into training and testing sets, ensuring there was enough labelled data to train and test with on a shallow network – of which there was a significant lack, and ensuring the datasets covered a wide range of chords that I would deem acceptable. Upon meeting with Li Zhang earlier in term 2, my expectations were managed when it became apparent that attempting to classify over 20+ different chord types was not feasible and instead limiting the problem to 10 or less would help me refine the network and understand what my code was doing.

Finally, I had issues setting up my local environment to work with Jupyter Notebook and spent almost 2 weeks equivalent of work hours attempting to fix this. In hindsight I should have started setting up my local machine to run my code in term 1, however I opted to rely on Google Colabatory. This was not a bad decision as Google Colabatory is the standard to create AI models and experiments with. My issue presented itself in data cleaning and preparation, which if using large datasets becomes tedious to repeatedly upload to Google Colabatory since its online only and their free session have a time limit, and if the time limit is exceeded all uploaded files need to be reuploaded. In summary, future planning and testing with Jupyter Notebook would have significantly sped up my project creation.

Chapter 8: Diary Log

8.1 October

This month I spent most of my time analysing and researching the broad area of chord classification. I also spent time looking at my old implementation in sixth form that used a CNN from scratch and used image classification.

I've completed my project plan by encompassing elements I have learnt from prior experiences of guitar and music theory, as well as reading introductory materials into the realm of guitar chord recognition. Such examples include PCPs, Discrete Fourier Transforms and Constant Q Transforms.

At this point I discovered that most audio classifications use CNNs to some degree, either via directly inputting an array or by using images so I might start investigating the use of CNNs again and see what comes out of it. I also discovered that some researchers attempted to use HMMs to detect and predict progressions.

I have made the decision to research some ML frameworks for displaying your work, such as Gradio, which enables you to embed TensorFlow to a GUI. This should make telling users the chord they've played much simpler and user friendly.

8.2 November

This month has been a theory heavy month. I've read approximately 15+ articles or journals surrounding chord recognition. Large amounts draw on already pre-existing methods by Fujishima, either utilising PCPs, Chromas or Spectrograms. I also researched more into deep neural networks as a possibility which would require me to train from scratch by converting all datasets that I have into spectrograms and then feeding it into a network.

I also started looking at pre-existing chord recognition software and found that there are a few websites that will tell you the chords of a YouTube song. I have been trying to investigate these websites to see if there's any code about or at least get a general idea of how it is done.

After researching on the websites, I concluded that it is almost impossible to obtain any form of code the websites use, although I was expecting this. So, I turned my attention towards apps instead like chordai which supposedly detect what chord you are playing on your phone live. It is an iPhone App so I will most likely need to borrow a housemate's phone.

After experimenting with my housemates phone, I came to the conclusion that the layout, design and aesthetic of the app is in line with my ideas, so I will use it as a reference point for future.

I now must also start planning my interim report as I have spent a lot of time just researching and reading, I might not have ample time to properly do a fully-fledged proof of concept and may divert my efforts to a more theoretical piece.

8.3 December

I submitted my interim report, it unfortunately did not have an audio proof of concept however, it has an image recognition example using an example algorithm that I may end up using for the final project.

After meeting with supervisor I have come to realise the project is significantly harder to achieve good results than I previously thought, so I will need to go back to my plan and understand what needs to be changed.

The main areas I identified that needed to change was the focus from K-means, N-Grams and CNNs to transfer learning. Transfer learning in principle is taking an already existing model and applying the same weights to your current model. Given that I am immensely struggling to find a suitable database, I may have to switch all my current efforts to understanding this.

For now, I will return to finding datasets for guitar that are extensive enough for my project.

8.4 January

I found a dataset called GuitarSet which seems to be very extensive and detailed, my only issue is I can't seem to find much code online that details how to use this dataset. I have seen the dataset be mentioned in a few papers but not much actual code implementation. I suppose the issue is also that I do not fully grasp the idea of transfer learning just yet, but slowly the more I read papers the more I understand.

I have had another meeting with my supervisor where we discussed different algorithms for audio classification based on Pytorch and deep learning. I also got given a dataset called the ESC50 dataset which I will look at as it seems quite promising.

Upon researching the ESC50 dataset further, I have discovered a transfer learning example called YAMNet created by TensorFlow that looks to use a pretrained model on an image dataset and provides a tutorial of using it with the ESC50 dataset. I will study and understand this model as this seems promising.

8.5 February

I've been very busy recently and have not had much time during the early stages of the month, I however, had a meeting with my supervisor where we went over the ESC50 dataset and introduced how I can search for models and so forth.

It is drawing nearer to the date of the final submission so all my efforts have now shifted to getting some guitar datasets.

I found some guitar datasets, but they are quite tricky to deal with and took me around 2 weeks to fully understand how to transform data and clean data before attempting to feed it into a neural net. This was also combined with me learning TensorFlow since I have never used it prior to this project, so there was a lot of searching stack overflow and so forth, but I achieved it in the end.

8.6 March

This now the final month for submission, I feel confident on my theory and understanding although I fear I lack in the programming aspects. Currently, my data pre processing and cleaning is going well and I understand fully how TensorFlow represents Tensors and NumPy arrays.

I have built a normal proof of concept using ESC50 and IDMT-CHORDS-SMT on the YAMNet model although I am getting very poor results, I will most likely ask my supervisor what the issues could be.

After discussing with my supervisor, we came to the conclusion that I should look at more models or at the very least use more datasets to create a comparison between datasets to demonstrate that the model indeed does work, but there are issues with the data.

Now, I focus on my final stretch for the report and code.

Chapter 9: **Appendix**

9.1 Original Specification

Aims:

- To detect live guitar chords and recommend chord progressions.
- To understand machine learning algorithms for sound classification.

Background:

As a guitarist – be it fingerstyle, flatpicking or rhythm - it is fundamental that you learn music theory to a certain degree. Most importantly, chord theory. Despite this, learning guitar chord theory is not an easy task, especially when using alternative tunings, and often many guitarists neglect doing this.

Composition also heavily draws on chord theory, and without a solid foundation it becomes a tedious task.

I am to create a program in my desired language that will aid guitarists in understanding chord theory and progressions.

This will be achieved through:

- Sound classification using machine learning algorithms.
- Graphically illustrating chords that the user plays at any given time with relation to key signatures.
- Graphically illustrating chords that a user could potentially use based on previous classifications.

Early Deliverables

1. Report: Analysis on different machine learning algorithms regarding sound classification.
2. Report: Analysis of existing literature surrounding chord recognition and implementations.
    a. At least 7 journals or papers.
3. Demo: Program that illustrates some of the machine learning algorithms in report 1.
    a. Said algorithms will be tested on artificial data sets.
4. Report: Rough plan of intended implementation with justification.

Final Deliverables

7. The program must have a full object-oriented design.
8. The program will have a GUI that will be used to develop the sequences of chords and potential chords.
9. The program needs to be able to recognise chords directly from an audio input.

10. The program needs to be able to use previous classifications to determine the best next chords.
11. The program should be as efficient as possible.
12. Full report
    a. Detailed explanation of techniques used on the project.
    b. Discuss the choice of sound classification algorithm.
    c. Evaluate the accuracy of machine learning algorithm/s used.
    d. Evaluate results alongside issues faced and tackled.
    e. Include documentation of the code alongside comments and justifications.
    f. Review of the literature used to generate final program.
    g. Evaluation of accuracy of algorithms used via a variety of researched techniques.

9.2 Diagrams & Figures

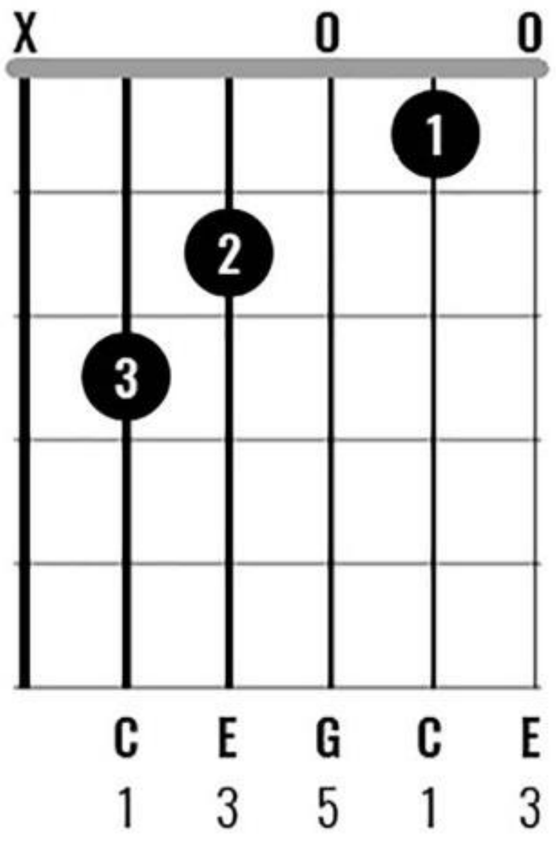


Figure 1 - C chord in Standard Tuning - https://guitar.com/lessons/beginner/learn-to-play-10-interesting-c-major-chord-variations/

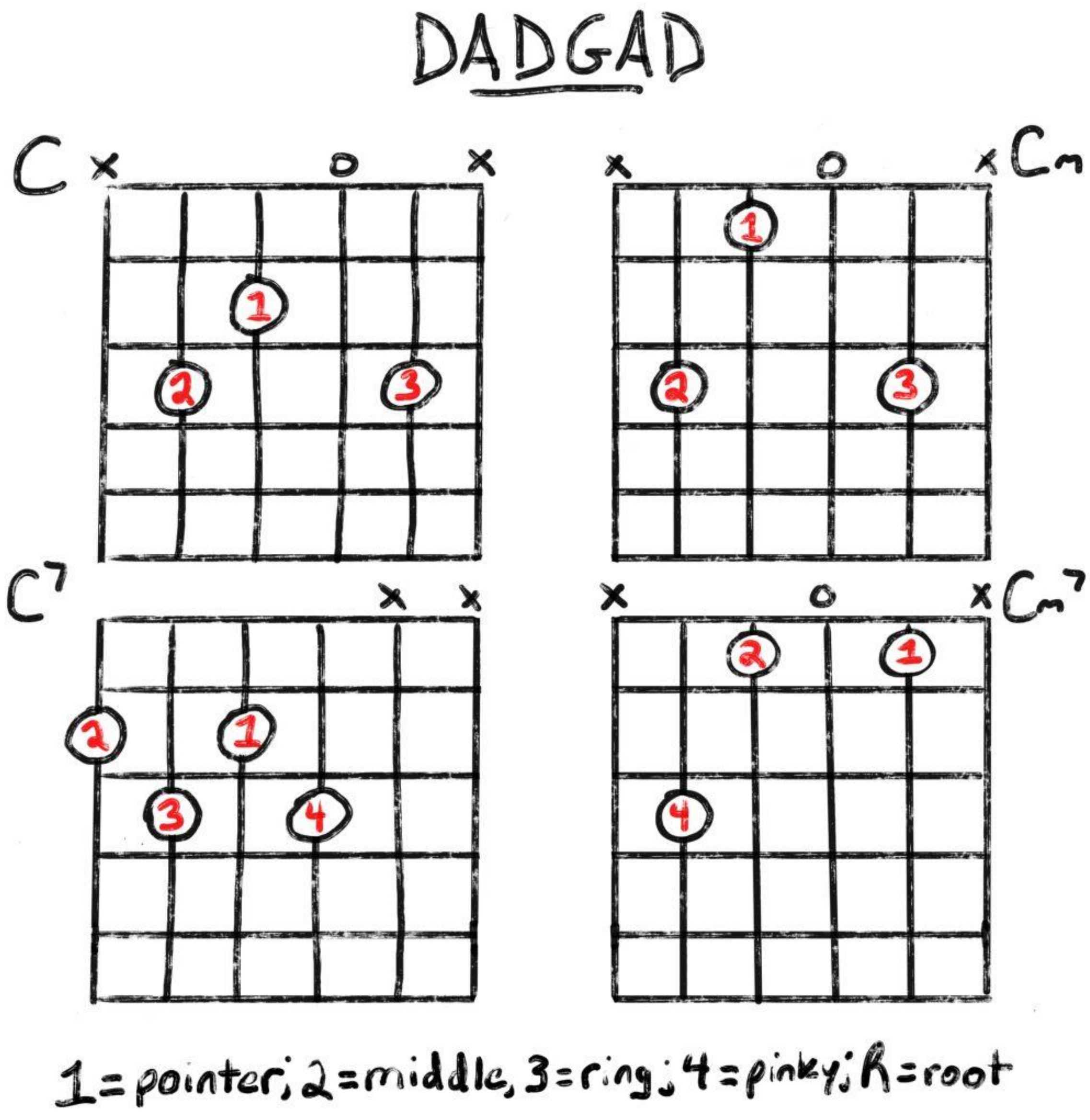


Figure 2 - C chord shapes in DADGAD tuning - https://growguitar.com/dadgad-chords/

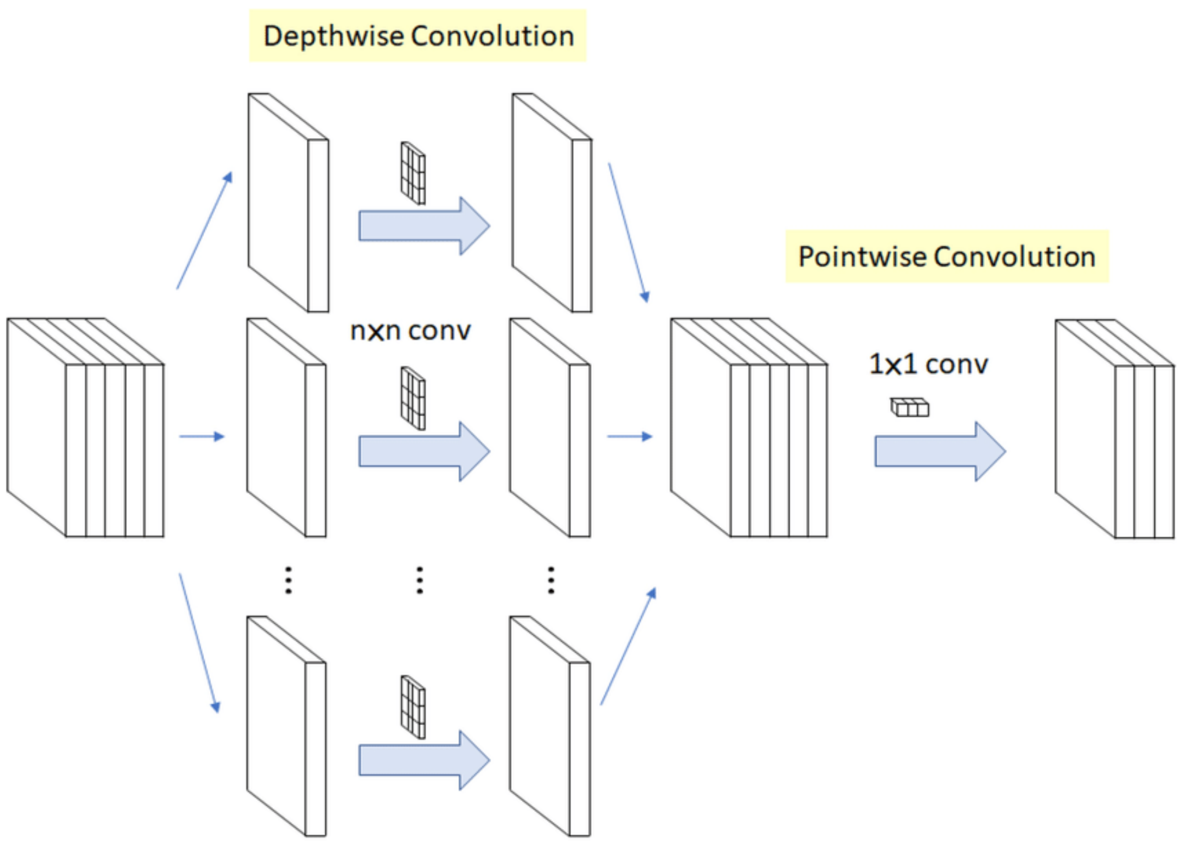
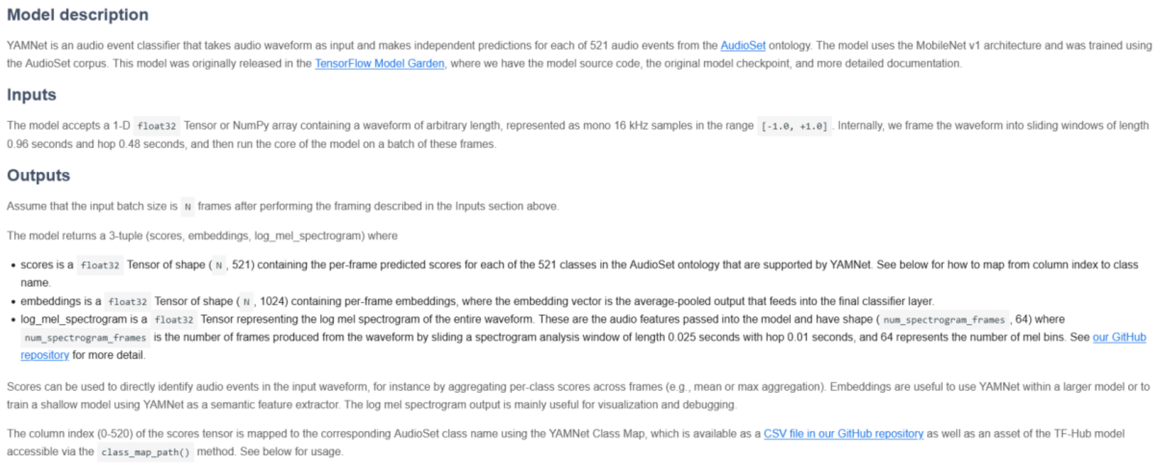


Figure 3 - Depth-wise Separable-Convolution [26]



Model description

YAMNet is an audio event classifier that takes audio waveform as input and makes independent predictions for each of 521 audio events from the AudioSet ontology. The model uses the MobileNet v1 architecture and was trained using the AudioSet corpus. This model was originally released in the TensorFlow Model Garden, where we have the model source code, the original model checkpoint, and more detailed documentation.

Inputs

The model accepts a 1-D `float32` Tensor or NumPy array containing a waveform of arbitrary length, represented as mono 16 kHz samples in the range `[-1.0, +1.0]`. Internally, we frame the waveform into sliding windows of length 0.96 seconds and hop 0.48 seconds, and then run the core of the model on a batch of these frames.

Outputs

Assume that the input batch size is `N` frames after performing the framing described in the Inputs section above.

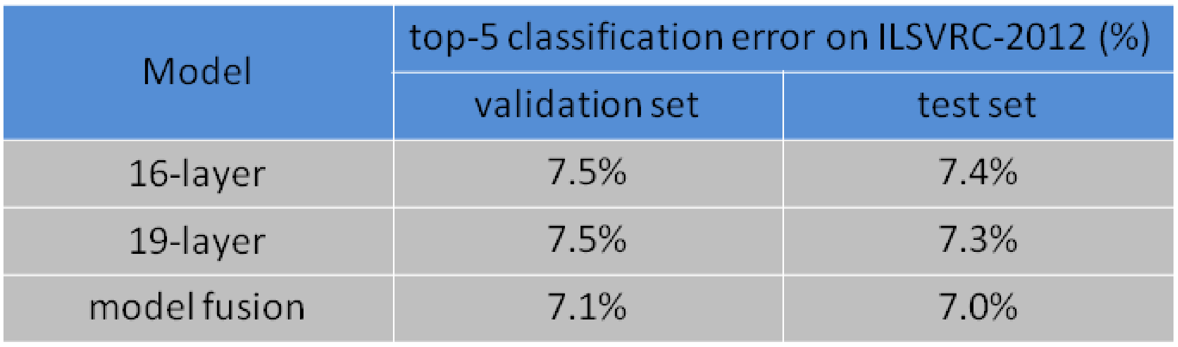
The model returns a 3-tuple (scores, embeddings, log_mel_spectrogram) where

- scores is a `float32` Tensor of shape (`N` , 521) containing the per-frame predicted scores for each of the 521 classes in the AudioSet ontology that are supported by YAMNet. See below for how to map from column index to class name.
- embeddings is a `float32` Tensor of shape (`N` , 1024) containing per-frame embeddings, where the embedding vector is the average-pooled output that feeds into the final classifier layer.
- log_mel_spectrogram is a `float32` Tensor representing the log mel spectrogram of the entire waveform. These are the audio features passed into the model and have shape (`num_spectrogram_frames` , 64) where `num_spectrogram_frames` is the number of frames produced from the waveform by sliding a spectrogram analysis window of length 0.025 seconds with hop 0.01 seconds, and 64 represents the number of mel bins. See our GitHub repository for more detail.

Scores can be used to directly identify audio events in the input waveform, for instance by aggregating per-class scores across frames (e.g., mean or max aggregation). Embeddings are useful to use YAMNet within a larger model or to train a shallow model using YAMNet as a semantic feature extractor. The log mel spectrogram output is mainly useful for visualization and debugging.

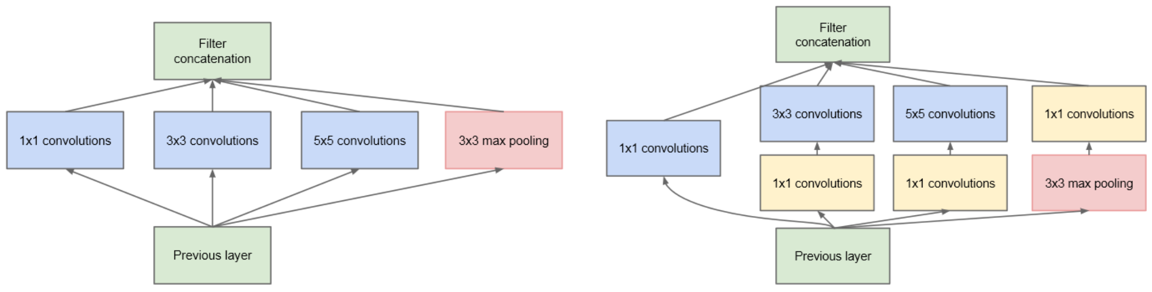
The column index (0-520) of the scores tensor is mapped to the corresponding AudioSet class name using the YAMNet Class Map, which is available as a CSV file in our GitHub repository as well as an asset of the TF-Hub model accessible via the `class_map_path()` method. See below for usage.

Figure 4 - TensorFlow YAMNet Model Description



Model	top-5 classification error on ILSVRC-2012 (%)	
	validation set	test set
16-layer	7.5%	7.4%
19-layer	7.5%	7.3%
model fusion	7.1%	7.0%

Figure 5 - ImageNet Challenge - VGG16 - https://www.robots.ox.ac.uk/~vgg/research/very_deep/



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 6 - GoogLeNet - Inception Modules - [7]

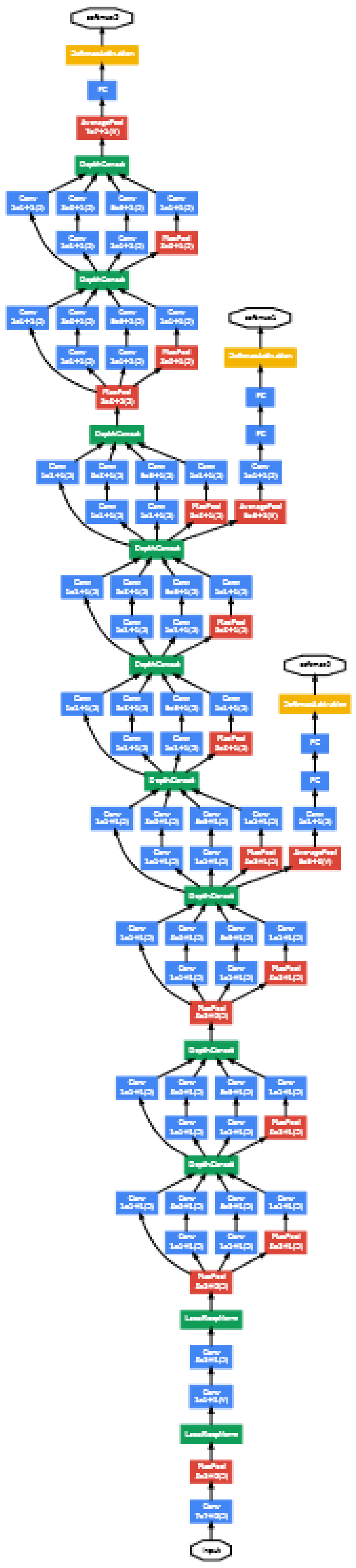


Figure 7 - Full GoogLeNet Network - [6]