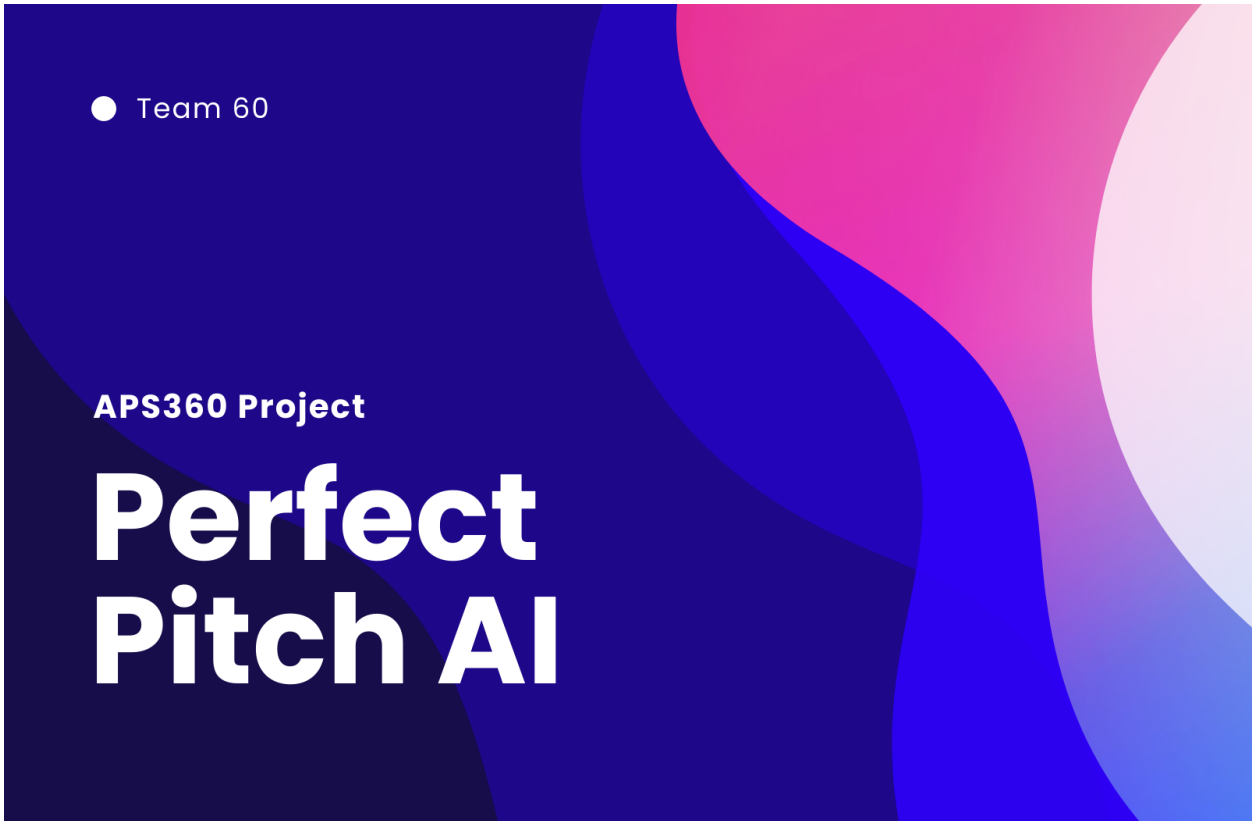
APS360 Final Report

Word Count: 2443



1.0 INTRODUCTION

Imagine you are in a coffee shop, and suddenly you hear a song that hits your sweet spot. This happens in many people's daily lives, including ours. Therefore, we are motivated to make music identification easier for music enthusiasts like ourselves.

Our goal is to create a tool that can quickly and precisely recognize any note played by any instrument, we call it the "Perfect Pitch AI". Our team believes that our model has many different potential applications. People may utilize our project by inputting several hundreds of snapshots of a song, and they will receive the overall note distribution of the song, and since it's hardly ever possible for two different songs to have the same sequence of notes, this data may be further used for training, for example, a song-recognition model. Google's song search by humming is another idea [2].

Machine learning is a reasonable approach because this can be seen as a classification problem. Different notes have different sounds that can generate different sound-wave diagrams which then can be used to train/test our model. It's similar to identifying gestures, and although these "musical gestures" are more complex and are harder to visualize, it's the same process of letting the neural network see multiple snapshots of different notes and generalize the overall distribution and curve of different wave diagrams.

Our model will be classifying 88 pitches from pitch 21 to pitch 108 of the MIDI standard from isolated samples of notes.

2.0 ILLUSTRATION/FIGURE

Figure 2.1: Illustration of the overall process

3.0 BACKGROUND/RELATED WORK

3.1 Background

Music consists of 12 distinct pitch classes(A A# B C C# D D# E F F# G G#) that are separated by their frequencies [1].

For example, the frequency of an A4 note is 440 Hz. However, there are many A notes, such as A5 note at 880 Hz and A3 note at 220 Hz. They sound similar and their waveforms look similar.

3.2 Related Work

One related application is tuning software. Tuning software analyzes the waveforms of the sound. They sample the instrument over a window of time and perform a fourier transformation

on the waveform to get a frequency amplitude function of the samples. Since there may be noise surrounding the note, the frequency amplitude plot won't be just a single frequency. Instruments don't produce a single frequency but a multitude of frequencies (aka harmonics). The fourier transformation is transformed again through the Harmonic Product Spectrum, which estimates the fundamental frequency (the peak with the lowest frequency) or the note itself [3].

Another application is called chordify.net. It finds the chords and the beats at which they occur at. They use deep learning for both beat recognition and chord recognition, where chords are defined by their fundamental pitch [5]. They used spectrograms of songs labelled with chords to train their models.

4.0 DATA PROCESSING

4.1 Data Metrics

The data set that we use is called NSynth, a high quality set of isolated, 4-second music notes samples annotated and recorded by Google's machine learning team Magenta 🎶 [6]. The original data set contains 1006 instruments, categorized by instrument family. The pitch distribution of the original data set is already imbalanced. As a result, we chose to take 1800 random samples from pitch 21 to pitch 108, to ensure a balanced distribution.

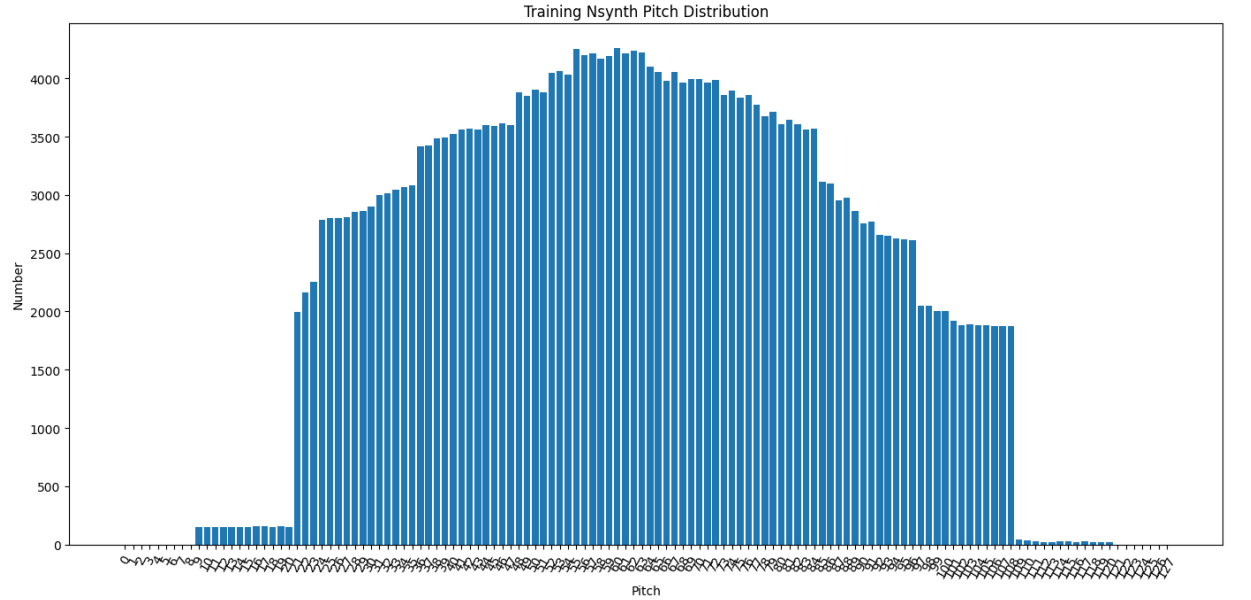


Figure 4.1.1: NSynth pitch distribution is unbalanced

NSynth and the used set's instrument distribution is imbalanced. The team did not take special care in balancing the instrument family because we hoped the model would be able to learn a

pitch embedding disregarding the instrument itself. The length at which the note was held also varied with time, but was stated to end at around 3 seconds generally.

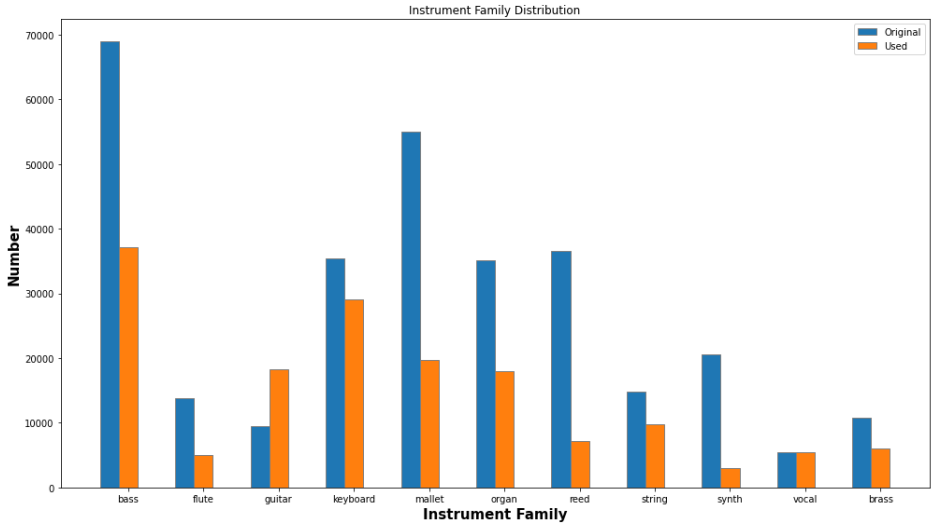


Figure 4.1.2: Training set instrument distribution (Blue = NSynth, Orange = Used for Training)

4.2 Preprocessing

We took a spectrogram of each sample. A spectrogram measures the strength of each frequency at point and time, making it similar to a heap map. Images were generated using Librosa, by converting the .wav files into spectrograms using Short-Term Fourier Transform (STFT) [8]. Amplitudes were then converted to decibel scale. Images are resized to 224 x 224 pixels to match AlexNet.

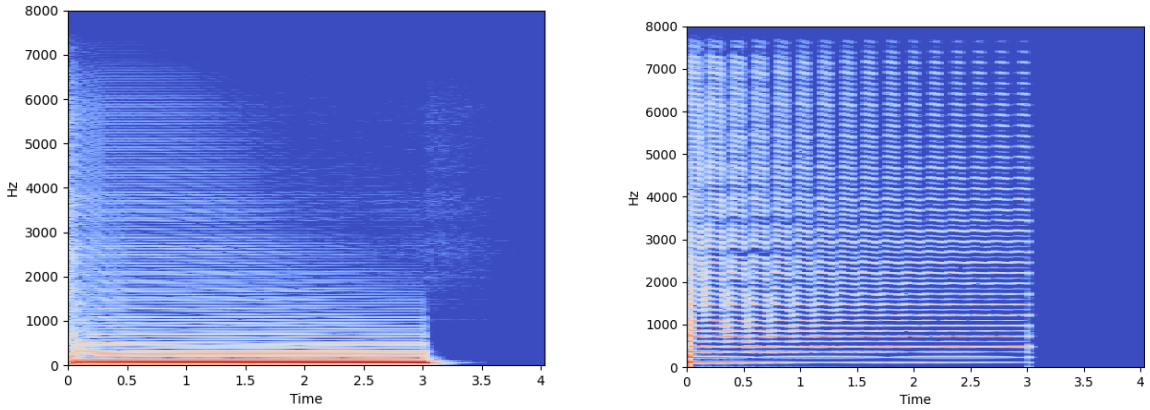


Figure 4.2.1: Spectrograms of C7 (Left) and B5 (Right).

4.3 Data Split

NSynth's validation set was used directly for validation (12732 samples).
NSynth's test set was used directly for testing (4293 samples).

Set	# Samples	%
Training	158400, 1800 each pitch	90%
Validation	12732	7.25%
Test	4293	2.75%

Table 1: Dataset Split used

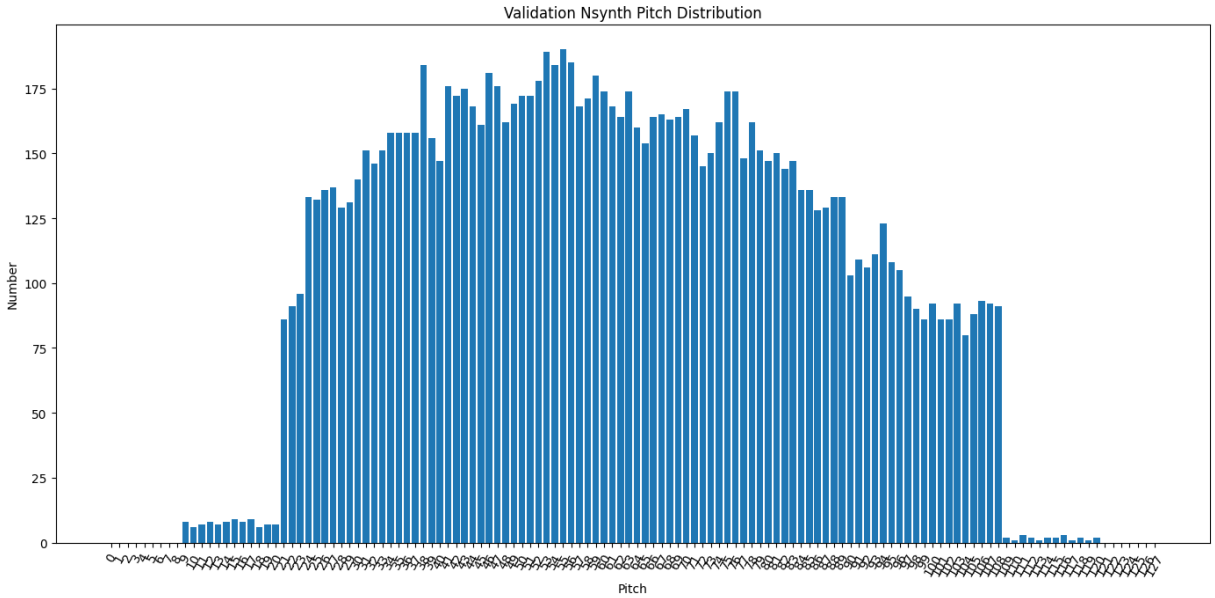


Figure 4.3.1: Pitch distribution seen in validation set

5.0 ARCHITECTURE

The primary model accepts images of dimensions 3x224x224. The image is passed through alexnet.features for feature extraction, then passed through to fully-connected layers (fig X). It was trained on an initial batch size of 188, learning rate of 1e-5, and 18 epochs. We multiplied the batch size by 1.1 times for every epoch loop after epoch 5. This acts like a various learning rate where at the beginning we want noisy gradients to jump out of local minimums and at the end we want gradients to be more accurate so that we can converge to the minimum.

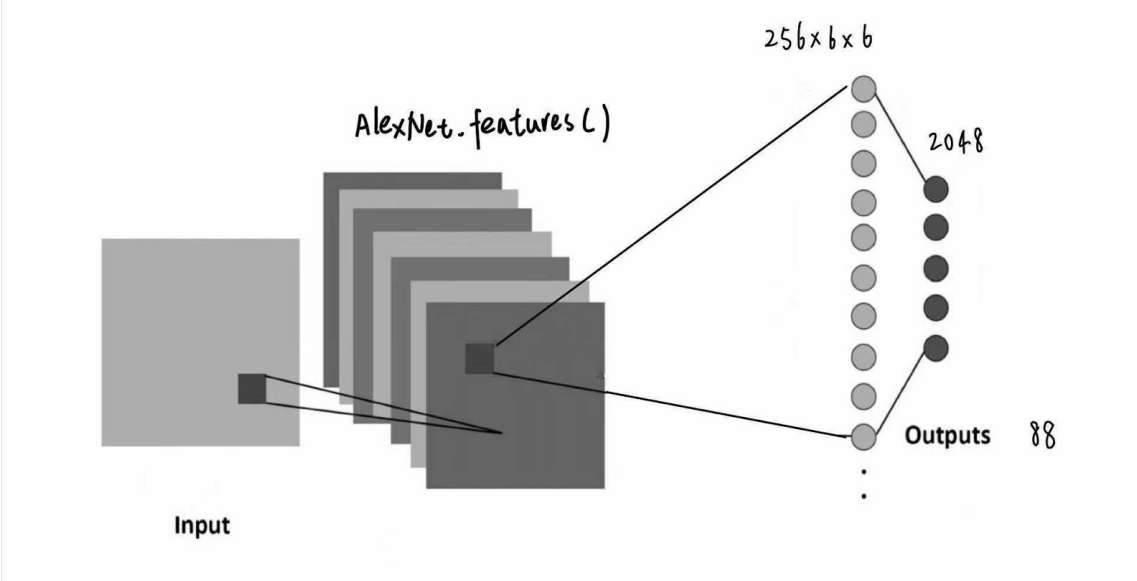


Figure 5.1: Primary Model Architecture

6.0 BASELINE MODEL

Choosing a reasonable baseline model is an important step in a machine learning problem. After we have done a lot of research on choosing the best and most suitable baseline model for our project and comparing the pros and cons of each doable baseline model. Our group shortlisted ANN,CNN and random forest classifier as baseline model choices. We decided to choose a random forest classifier as our final baseline model since it works well with a large amount of data and our project can be interpreted as a classification problem.

Random forest classifier is a method of classification. It can construct different trees at training time, and output the regression of individual trees [9]. After adjusting the number of estimators to 200, which is the number of total trees, the max-depth to 6, which is the maximum number of branches in each tree, the minimum sample split to 10, which can create arbitrary small leaves. The baseline model can achieve a training accuracy of 42.73%, which is a solid number for the baseline model.

We have also created a CNN baseline model without optimization and parameters tuning. This model can only achieve an overall accuracy of 37.8% at its best. Therefore, our group concluded that a random forest classifier is suitable for our project since it uses an ensemble method to construct multiple decision making trees.

7.0 QUANTITATIVE RESULT

Our model achieves 92.4% training accuracy, 82.8% validation accuracy and 80.4% testing accuracy. The model took over 10 hours to train for 18 epochs, and google-colab kept disconnecting before the model was fully trained. Therefore, the team could not obtain the whole training curve for the model. However, the validation loss dropped to a minimum at epoch 18 while the accuracy was the highest, and thus the team decided to use the model that was trained for 18 epochs. Since the purpose of the model is to identify different pitches, the team balanced the data set in terms of pitch. However, we recognized that the data that we use varied in many areas which made the prediction much more challenging. For example, the data varied in instruments, it also varied in instrument families. For example, keyboards can be acoustic, electronic or synthetic. The data also varied in length. Some samples contain three second of silence and only one second of actual sound. Therefore, the team decided to try testing with some tolerance. The tolerance would be three pitches, which means the model predicted pitch 70 and the actual pitch is 67, we considered the model to be correct. It turned out the testing accuracy with tolerance increased significantly to 88.7% which shows that the model is predicting on the right track and the predictions are reasonable and logical.

8.0 QUALITATIVE RESULT

Model performed worse on lower pitch notes. We believe that this is due to the overlapping of tones, as pitches become lower, the strings required to play them become thicker, and many overtones can become dominant in the sound and overwhelm the fundamental tone.[10]

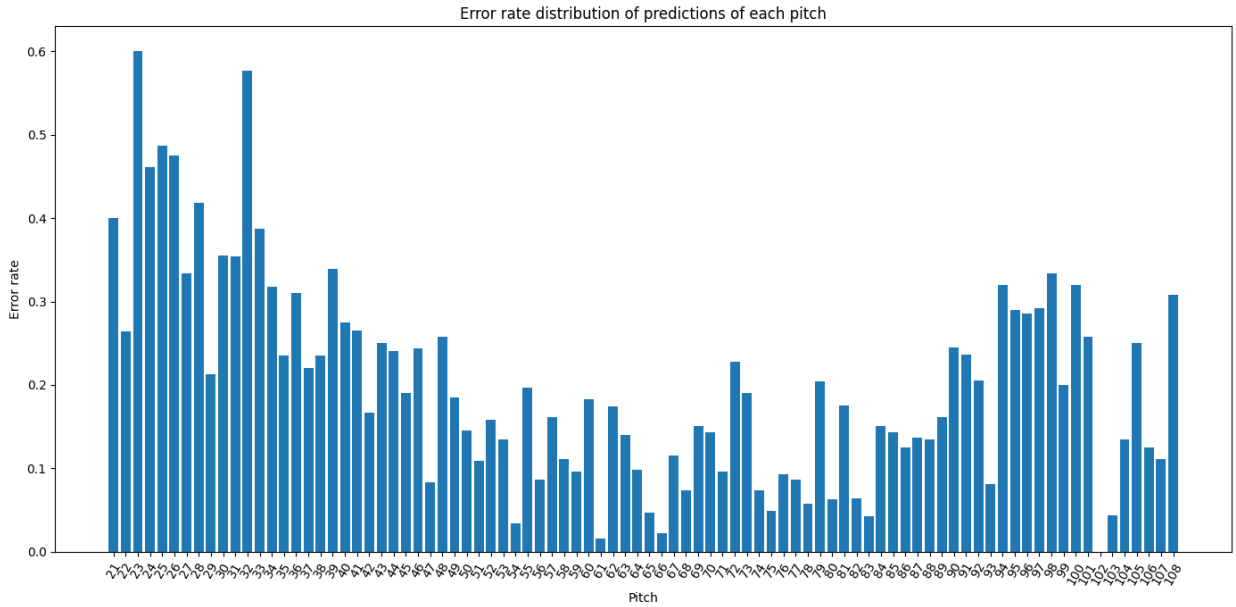


Figure 8.1: Error rate distribution by pitch

Another result is that the model performed worse on certain instruments. This is due to the imbalance of the data set in terms of instrument and instrument family. For example, the training set contained only under 10000 flute samples while it contained over 60000 bass samples.

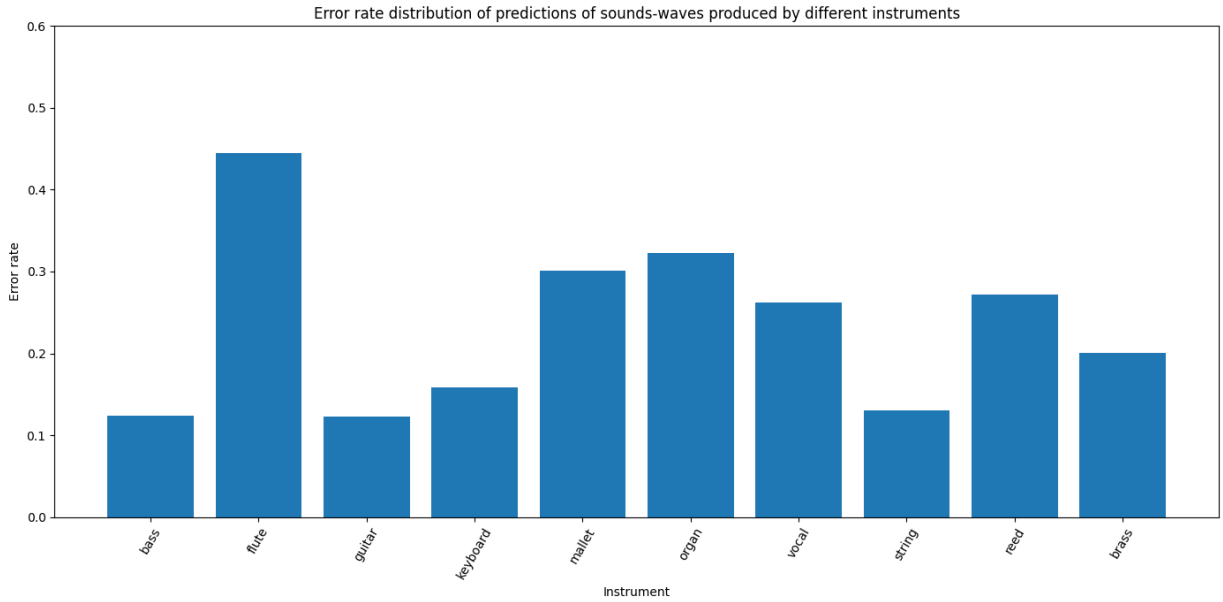


Figure 8.2: Error rate distribution by instrument

9.0 EVALUATE MODEL ON NEW DATA

9.1 Applying Model to real life samples

To evaluate the model, we first recorded audio samples of three different pitches, using an online virtual keyboard. Each audio sample lasts for approximately four seconds, same as the samples in the NSynth data set. Then we feed these audio samples into a program that we have written for transforming .wav files to spectrograms. After we have obtained the newly generated spectrograms, we fed them into our model to see if our model can correctly predict all the labels. The correct labels for each pitch could be found using the MIDI chart.

9.2 Applying Model to another large dataset

9.2.1 Dataset Description

TinySOL dataset was used, containing 2478 samples recorded in 1990's of various instruments [7]. Most instrument classes are a part of NSynth however, specific instruments aren't specified in NSynth. Please note that the instruments such as Accordion are not found within the

instrument classes of NSynth. And the dataset also includes varying lengths of data (from 3s - 8s), thus making it a good choice for testing on the model's ability to generalize.

1. Bass Tuba
2. French Horn
3. Trombone
4. Trumpet in C
5. Accordion
6. Contrabass
7. Violin
8. Viola
9. Violoncello
10. Bassoon
11. Clarinet in B-flat
12. Flute
13. Oboe
14. Alto Saxophone

List 1: List of Instruments

9.2.2 Performance on TinySol

Our model obtained a very low accuracy on this dataset of about 1%. This illustrates the limitations of our model. The data is varying in length and also has a different sampling rate. Moreover, the data is very noisy as it was recorded a long time ago. All of which are aspects that our model can improve on in the future.

10.0 DISCUSSION

We believe our model is performing well overall, as training, validation and test accuracies are over 80%.

What surprised us was the model's ability to differentiate different spectrograms at such high accuracy. Some spectrograms of the same pitch can look extremely different due to length (some samples have only 1 second of sound with 3 seconds of silence), instrument and instrument family. Interestingly, it seems that although our model has some limitations, it is doing overall very well across a wide range of instruments and data-sample lengths. We were also surprised that the model could accurately predict 88 highly specific notes, as our initial idea was to train it to generalize only 12 fundamental notes.

It also seems from testing on the TinySOL data that further improvements could be made on the data processing of the images. Possible ideas would be to take the color channels directly instead of relying on the library plots and sample the STFT at a constant number of points [4]. One reasoning of why the model may have failed to work on the TinySOL data would be that the axises of the plots were different magnitudes, thus possibly the model had not learnt that reading the axis was important, especially since it was uniform for NSynth data.

Our model has room for improvement:

- The inputs are all 4-second long, which could make our model inflexible. Choosing a more varied dataset or sampling a fixed number of STFT points could make it more generalized [8].
- NSynth data lacks noise in the data, since they are very high quality, which could affect our model's performance in the real world since data inputted into our model is often noisy. Adding noise to random samples may help learn classification with noise.
- Choosing a larger input feature such as a spectrogram with metadata properties found in NSynth's description. Data such as velocities and pitch characteristics may help classification.

Overall, we learned that machine learning models are not easy to construct, and in the real world, pre-constructed models like in the labs will often not be given to us, and this experience helped us conceptualize on how to use different models to solve different tasks. Moreover, after watching the final presentations, we learned the vast variety of problems that machine-learning can solve, and it has largely increased our interest towards machine learning.

11.0 ETHICAL CONSIDERATION

- Copyright issues may arise if we choose to record the sound sources from existing apps. Therefore, we will conduct carefully when choosing external recording resources.
- Musicians may not want their music to be transcribed without their permission. Like all artists they may wish to be unique and software like this takes away their choice of sharing their art.

12.0 PROJECT DIFFICULTY

We believe that what we achieved is intermediate in difficulty, but to further perfect the model would be very challenging.

The hardest part of our project is the data preprocessing part. It was not trivial to think about an effective method to convert .wav files to a specific format data which can be loaded into a

dataset. Additionally, some of the noises during the process of transferring still exist in our training data and it is challenging to completely filter out those unexpected noises. Another aspect that was demanding was finding a suitable model for such a diverse dataset, and the model needed to accurately predict 88 classes. Many models were tried including different CNN architectures and different transfer learning architectures such as resnet.

The most challenging part of the project going on into the future is to extend the model to classify music and not just isolated notes. Most songs are composed of melody, bass and harmony, often with a variety of voices and instruments being played at a time. This means the model will need to determine which note to classify, which really depends on the application. This also means ensuring the model learns to classify with noise such as static or random background noises. Either way, our model serves as a good base to further explore different applications.

13.0 Works Cited

1. M. Bourne, “What are the frequencies of music notes?” [Online]. Available: https://www.intmath.com/trigonometric-graphs/music.php. [Accessed: 13-Feb-2021].
2. M. Deshpande, “Musical Pitch Identification.” .
3. Naiadseye, “Fast Fourier Transform Tuner,” naiadseye, 09-Oct-2013. [Online]. Available: https://naiadseye.wordpress.com/2013/10/09/fast-fourier-transform-tuner/. [Accessed: 13-Feb-2021].
4. “Short-time Fourier transform,” Wikipedia, 28-Dec-2020. [Online]. Available: https://en.wikipedia.org/wiki/Short-time_Fourier_transform. [Accessed: 13-Feb-2021].
5. J. 06, “What is Chordify?,” *Chordify Support*. [Online]. Available: https://support.chordify.net/hc/en-us/articles/360002221018-What-is-Chordify-#:~:text=Chordify%20is%20an%20online%20music,a%20simple%20and%20intuitive%20player. [Accessed: 13-Feb-2021].
6. J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi, “Neural audio synthesis of musical notes with WAVENET AUTOENCODERS,” 05-Apr-2017. [Online]. Available: https://arxiv.org/abs/1704.01279. [Accessed: 10-Apr-2021].
7. C. Emanuele, D. Ghisi, V. Lostanlen, F. Lévy, J. Finebergand Y. Maresz, “TinySOL: an audio dataset of isolated musical notes”. Zenodo, 31-Jan-2020, doi: 10.5281/zenodo.3633012.
8. “librosa.stft¶,” *librosa.stft - librosa 0.8.0 documentation*. [Online]. Available: https://librosa.org/doc/0.8.0/generated/librosa.stft.html. [Accessed: 10-Apr-2021].
9. Sklearn.ensemble.randomforestclassifier¶. (n.d.). From https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
10. https://www.quora.com/Can-people-with-perfect-pitch-identify-notes-of-very-high-or-low-frequencies

Appendix A: Colab Link and Code Snippets

Colab contains only training code, other snippets include data preprocessing.

Colab link:
https://colab.research.google.com/drive/1IQp98412QZzDXL-zVpnVYCsKhSxWpeZ5?usp=sharing

Data Preprocessing Code:

```
import numpy as np
import librosa
import os.path
import json
import torch
import torch.nn as nn
import torch.nn.functional as F
import librosa.display
import matplotlib.pyplot as plt
from matplotlib.backends.backend_agg import FigureCanvasAgg as
FigureCanvas
import skimage.io
NUM_POINTS = 100
TOTAL_NUM_PITCHES = 128
DOT_WAV = '.wav'
def written_note_to_pitch(note):
  switcher = {
    "C": 0,
    "C#": 1,
    "D": 2,
    "D#": 3,
    "E": 4,
    "F": 5,
    "F#":6,
    "G": 7,
    "G#":8,
    "A": 9,
    "A#":10,
    "B": 11,
  }
  octave = -1
  writtenNote = ""
```

```
  if note[1] == "#":
    writtenNote = note[:2]
    octave = int(note[2])
  else:
    writtenNote = note[0]
    octave = int(note[1])
  print(note,  octave*12 + switcher.get(writtenNote))
  return octave*12 + switcher.get(writtenNote)
class FileReader():
    def __init__(self):
        pass

    def read_entire(self, file_path):
      with open(file_path, 'r') as content_file:
        content = content_file.read()
        return content
class NSynthData():
  def __init__(self, path_to_file, metadata):
    self._path_to_file = path_to_file
    self._metadata = metadata
    self._time_series, self._sampling_rate = librosa.load(path_to_file,
sr=self._metadata['sample_rate'])
    print(self._time_series.shape)
  def get_name(self):
    return self._metadata['note_str']

  def get_pitch(self):
    return self._metadata['pitch'] % TOTAL_NUM_PITCHES

  def get_audio_as_time_series(self):
    return self._time_series

  def get_audio_sampling_rate(self):
    return self._sampling_rate
class TinySOLData():
  def __init__(self, path_to_file):
    self._path_to_file = path_to_file
    self._name = self._path_to_file.split("\\")[-1][:-4]
    print(self._name)
    self._pitch = written_note_to_pitch(self._name.split("-")[2])
```

```
    self._time_series, self._sampling_rate = librosa.load(path_to_file,
sr=44100)
  def get_name(self):
    return self._name

  def get_pitch(self):
    return self._pitch

  def get_audio_as_time_series(self):
    return self._time_series

  def get_audio_sampling_rate(self):
    return self._sampling_rate

class NSynthFormattedData():
  def __init__(self, data: NSynthData,
mel_frequency_cepstral_coefficients: np.array):
    self._name=data.get_name()
    self._pitch=data.get_pitch()
    self._mffcs=mel_frequency_cepstral_coefficients
  def get_formatted_data(self):
    return self._mffcs
  def get_name(self):
    return self._name
  def get_pitch(self):
    return self._pitch  % TOTAL_NUM_PITCHES
class TinySOLDataLoader():
  def __init__(self, path_to_audio_folder: str):
    self._path_to_audio_folder = path_to_audio_folder

  def __iter__(self):
    print(self._path_to_audio_folder)
    total_samples = len(os.listdir(self._path_to_audio_folder))
    for i, (path_to_individual_audio) in
enumerate(os.listdir(self._path_to_audio_folder)):
      path_to_file = os.path.join(self._path_to_audio_folder,
path_to_individual_audio)
      print("Progress: {}/{}".format(i, total_samples))
      newData = TinySOLData(path_to_file)
      yield newData
```

```
class NSynthDataLoader():
  def __init__(self, path_to_metadata: str, path_to_audio_folder: str):
    self._path_to_metadata = path_to_metadata
    self._path_to_audio_folder = path_to_audio_folder

    self._file_reader = FileReader()
    self._audio_metadata_as_str = self._file_reader.read_entire(
        path_to_metadata)
    self._audio_metadata_as_dict = json.loads(self._audio_metadata_as_str)

  def __iter__(self):
    total_samples = len(self._audio_metadata_as_dict)
    for i, (key, value) in
enumerate(self._audio_metadata_as_dict.items()):
      path_to_file = os.path.join(self._path_to_audio_folder, key) +
DOT_WAV
      print("Progress: {}/{}".format(i, total_samples))
      newData = NSynthData(path_to_file, value)
      yield newData


class NSynthDataWriter():
  def __init__(self, folderToWriteTo: str):
    self._folder = folderToWriteTo
    if not os.path.exists(folderToWriteTo):
      os.makedirs(folderToWriteTo)
    for i in range(TOTAL_NUM_PITCHES):
      inner_folder = os.path.join(folderToWriteTo, str(i))
      if not os.path.exists(inner_folder):
        os.makedirs(inner_folder)


  def write_data(self, data: NSynthFormattedData):
    filename = os.path.join(self._folder, str(data.get_pitch()),
data.get_name())
    torch.save(data.get_formatted_data(), filename)
class NSynthImageWriter():
  def __init__(self, folderToWriteTo: str):
    print(folderToWriteTo)
```

```
    self._folder = folderToWriteTo
    if not os.path.exists(folderToWriteTo):
      os.makedirs(folderToWriteTo)
    for i in range(TOTAL_NUM_PITCHES):
      inner_folder = os.path.join(folderToWriteTo, str(i))
      if not os.path.exists(inner_folder):
        os.makedirs(inner_folder)


  def write_data(self, data: NSynthData):
    filename = os.path.join(self._folder,
str(data.get_pitch()%TOTAL_NUM_PITCHES), data.get_name() + ".png")
    print(self._folder, str(data.get_pitch()%TOTAL_NUM_PITCHES),
data.get_name() + ".png")
    fig = plt.Figure()
    canvas = FigureCanvas(fig)
    ax = fig.add_subplot(111)


    X = librosa.stft(data.get_audio_as_time_series())
    Xdb = librosa.amplitude_to_db(abs(X))
    aok = librosa.display.specshow(Xdb, sr=data.get_audio_sampling_rate(),
ax= ax,x_axis='time', y_axis='hz')
    fig.savefig(filename)
    # skimage.io.imsave(filename, img)
    #
self.spectrogram_image(data.get_audio_as_time_series(),data.get_audio_samp
ling_rate(), filename, 512, 128)
  def scale_minmax(self, X, min=0.0, max=1.0):
      X_std = (X - X.min()) / (X.max() - X.min())
      X_scaled = X_std * (max - min) + min
      return X_scaled
  def spectrogram_image(self, y, sr, out, hop_length, n_mels):
      # use log-melspectrogram
      mels = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=n_mels,
                                            n_fft=hop_length*2,
hop_length=hop_length)
      mels = np.log(mels + 1e-9) # add small number to avoid log(0)

      # min-max scale to fit inside 8-bit range
```

```
      img = self.scale_minmax(mels, 0, 255).astype(np.uint8)
      img = np.flip(img, axis=0) # put low frequencies at the bottom in
image
      img = 255-img # invert. make black==more energy

      # save as PNG
      skimage.io.imsave(out, img)

class NSynthDataFormatter():
  def __init__(self):
    pass

  def format_data(self, data: NSynthData):
    y=data.get_audio_as_time_series()
    sr = data.get_audio_sampling_rate()
    chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
    zcr = librosa.feature.zero_crossing_rate(y)
    idx = np.round(np.linspace(0, zcr.shape[1] - 1,
NUM_POINTS)).astype(int)
    zcr = zcr[:,idx]
    features = np.mean(chroma_stft, axis=1)
    return NSynthFormattedData(data, torch.cat((torch.tensor(features),
torch.tensor(zcr[0]))))
    # y=data.get_audio_as_time_series()
    # sr = data.get_audio_sampling_rate()
    # X = librosa.stft(data.get_audio_as_time_series())
    # idx = np.round(np.linspace(0, X.shape[1] - 1,
NUM_POINTS)).astype(int)
    # X = X[:,idx]

class NSynthDataProcessor():
  def __init__(self, loader: NSynthDataLoader, formatter:
NSynthDataFormatter, writer: NSynthDataWriter, imageWriter:
NSynthImageWriter):
    self._loader = loader
    self._formatter = formatter
    self._writer = writer
    self._imgwriter = imageWriter

  def process(self, number_samples=1):
```

```
    for data in self._loader:
      # self._imgwriter.write_data(data)
      formatted_data = self._formatter.format_data(data)
      self._writer.write_data(formatted_data)


class NSynthDataCalculator():
  def __init__(self):
    pass
  def calculate_distribution_pitch(self, folder):
    mapping = {}
    for name in os.listdir(folder):
      mapping[name] = len(os.listdir(os.path.join(folder, name)))
    print(mapping)
    return mapping

  def calculate_distribution_instrument(self, folder):
    class_mapping = {}
    instrument_mapping = {}
    class_instrument_mapping = {}
    for name in os.listdir(folder):
      for file_name in os.listdir(os.path.join(folder, name)):
        class_name = file_name.split("_")[0]
        instrument_name = file_name.split("_")[1]
        class_instrument_name = class_name + "_" + instrument_name
        if class_name in class_mapping:
          class_mapping[class_name] += 1
        else:
          class_mapping[class_name] = 1

        if instrument_name in instrument_mapping:
          instrument_mapping[instrument_name] += 1
        else:
          instrument_mapping[instrument_name] = 1

        if class_instrument_name in class_instrument_mapping:
          class_instrument_mapping[class_instrument_name] += 1
        else:
          class_instrument_mapping[class_instrument_name] = 1
    return class_mapping, instrument_mapping, class_instrument_mapping
```

```
test_val_train = 'test'#test valid
loader=NSynthDataLoader('C:\\Users\\User\\Downloads\\nsynth-{}\\examples.j
son'.format(test_val_train),

'C:\\Users\\User\\Downloads\\nsynth-{}\\audio'.format(test_val_train))
imageWriter = NSynthImageWriter(os.path.join(test_val_train))
formatter=NSynthDataFormatter()
writer = NSynthDataWriter(os.path.join(test_val_train))
processor=NSynthDataProcessor(loader, formatter, writer, imageWriter)
formatted_samples=processor.process()
```

Baseline Model:

```
import pandas as pd
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score


train_feature = []
train_label = []
test_feature = []
test_label = []

for data,labels in train_loader:
  train_feature.append(data)
  train_label.append(labels)

train_feature = torch.stack(train_feature)
train_label = torch.stack(train_label)

n,i,j,k,c=train_feature.shape
train_feature=train_feature.reshape(n,i*j*k*c)

for data, labels in test_loader:
  test_feature.append(data)
  test_label.append(labels)

test_feature = torch.stack(test_feature)
test_label = torch.stack(test_label)

n,i,j,k,c=test_feature.shape
test_feature =test_feature.reshape(n,i*j*k*c)

baseline_model= RandomForestClassifier(bootstrap=True,
        min_samples_leaf=3,
        n_estimators=200,
        min_samples_split=10,
        max_features='auto',
        max_depth=6,
        max_leaf_nodes=None)

baseline_model.fit(train_feature,train_label.ravel())
predictions = baseline_model.predict(train_feature)
TrainingAccurary = accuracy_score(test_label,predictions)

print("accuracy of baseline model: {0}".TrainingAccurary)

```