Machine Learning Engineer Nanodegree

Capstone Project

Salih Güngör

June 1st, 2018

I. Definition

Project Overview

I'm working on a Kaggle Project that you can find on this link. Also, the data I use for this project can be found here along with the definitions. We usually experience a mix of sounds that create an atmosphere - like the sound of construction, a traffic noise coming from outside the door, a noisy laugh in the room and a passing clock on the wall. Due to the wide range of sounds we experience in part, reliable automated general purpose voice labeling systems are not available. At the moment there is a lot of manual effort to do tasks such as adding comments to audio collections and providing subtitles for non-speech events in audiovisual context. To solve this problem, I am developing a machine learning model to classify 41 audio sources as input as audio signals.

Problem Statement

The problem is to classify and understand the audio sources (analog audio data / raw data) in the audio signal (extracting useful features from audio signals). I am using the tagged dataset (pre-classified audio signals) for this problem, and I am preparing a CNN model to learn these classes. CNN classifies the untagged signal more than model and one of these 41 sources classifies the label in that signal. There may be other audio sources in my data. However, I am not interested in other untagged audio sources for this project. I'm just trying to label these 41 tags.

Metrics

My evaluation metrics will be F1 score. I used it in our previous projects. A good advantage of F1 score: If there are crooked classes like this one, for example there are 41 different labels, but some tags are in other tags. If I guess a cluster that is smaller than it in a not too big main cluster, I cannot say that I made it with the right guess. However, if the entire recording is accurate, this is an excellent prediction. If I make a mistaken guess in a small cluster, it may not reduce the accuracy. Because the number of values in the cluster is small and it will be difficult to correct.

In such situations, precision and recall become very useful. These two measurements can be combined to obtain the F1 score; this is the weighted average of the sensitivity and recall scores (harmonic mean). This score can range from 0 to 1, with 1 being the best possible F1 score (we get the harmonic average for the ratios).

II. Analysis

Data Exploration

Dataset and Inputs

You can find the used data on the kaggle page. (The data set is taken from kaggle)

I will extract some features using the Mel Frequency Cepstral Coefficient (MFCC) algorithm. The anchor basically divides into fragments and summarizes the data framed. For more details, see Algorithm, Application.

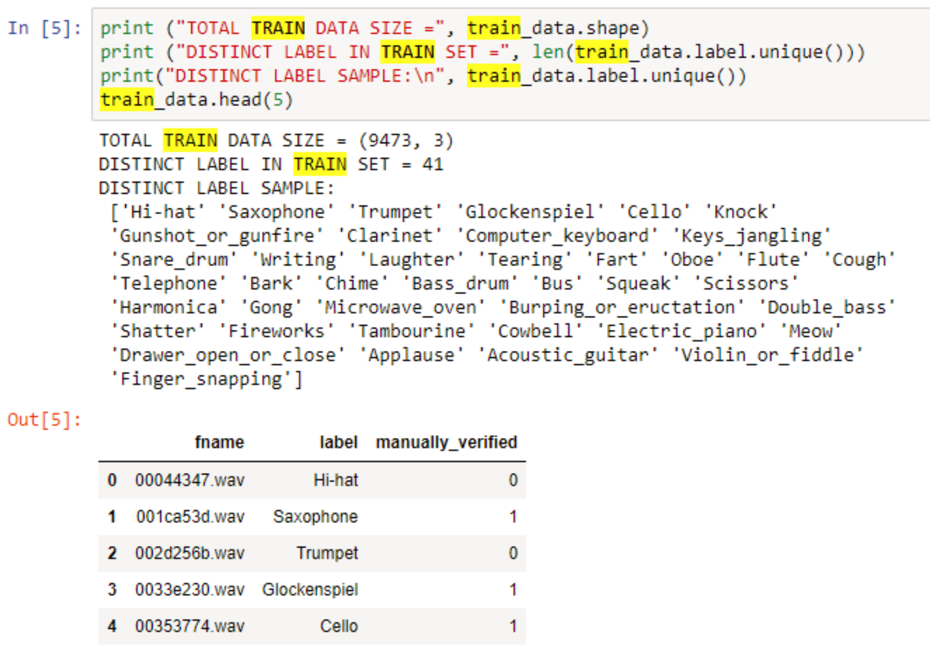
File Descriptions

- train.csv - ground truth labels and other information relating to the training audio
- audio_train.zip - a folder containing the audio (.wav) training files
- audio_test.zip - a folder containing the audio (.wav) test files
- sample_submission.csv - a sample submission file in the correct format; contains the list of files found in the audio_test.zip folder

Data Fields

Each row of the train.csv file contains the following information:

- fname: the file name
- label: the audio classification label (ground truth)
- manually_verified: Boolean (1 or 0) flag to indicate whether or not that annotation has been manually verified; see below for additional background.



```
In [5]: print ("TOTAL TRAIN DATA SIZE =", train_data.shape)
        print ("DISTINCT LABEL IN TRAIN SET =", len(train_data.label.unique()))
        print("DISTINCT LABEL SAMPLE:\n", train_data.label.unique())
        train_data.head(5)

TOTAL TRAIN DATA SIZE = (9473, 3)
DISTINCT LABEL IN TRAIN SET = 41
DISTINCT LABEL SAMPLE:
 ['Hi-hat' 'Saxophone' 'Trumpet' 'Glockenspiel' 'Cello' 'Knock'
 'Gunshot_or_gunfire' 'Clarinet' 'Computer_keyboard' 'Keys_jangling'
 'Snare_drum' 'Writing' 'Laughter' 'Tearing' 'Fart' 'Oboe' 'Flute' 'Cough'
 'Telephone' 'Bark' 'Chime' 'Bass_drum' 'Bus' 'Squeak' 'Scissors'
 'Harmonica' 'Gong' 'Microwave_oven' 'Burping_or_eructation' 'Double_bass'
 'Shatter' 'Fireworks' 'Tambourine' 'Cowbell' 'Electric_piano' 'Meow'
 'Drawer_open_or_close' 'Applause' 'Acoustic_guitar' 'Violin_or_fiddle'
 'Finger_snapping']

Out[5]:
```

	fname	label	manually_verified
0	00044347.wav	Hi-hat	0
1	001ca53d.wav	Saxophone	1
2	002d256b.wav	Trumpet	0
3	0033e230.wav	Glockenspiel	1
4	00353774.wav	Cello	1

About This Dataset (The data set is taken from kaggle)

Dataset Kaggle 2018 (or FSDKaggle2018 for short) is an audio dataset containing 18,873 audio files annotated with labels from Google's AudioSet Ontology.

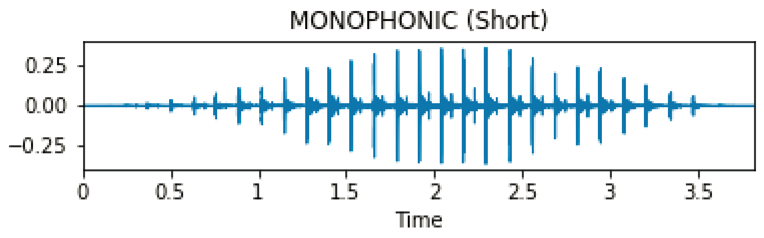
All audio samples in this dataset are gathered from Freesound and are provided here as uncompressed PCM 16 bit, 44.1 kHz, mono audio files. The ground truth data provided in this dataset has been obtained after a data labeling process which is described in the Data labeling process section below. FSDKaggle2018 sounds are unequally distributed in the following 41 categories of the AudioSet.

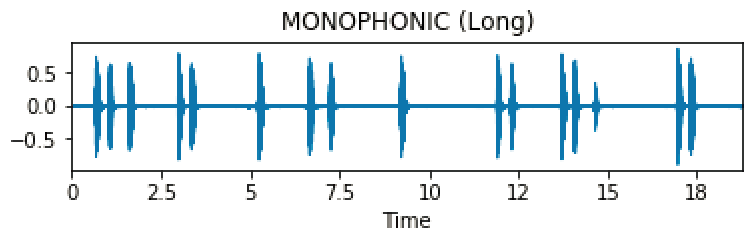
Ontology:

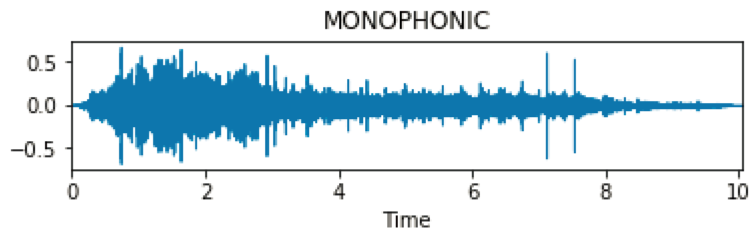
> “Acoustic_guitar”, “Applause”, “Bark”, “Bass_drum”, “Burping_or_eructation”, “Bus”, “Cello”, “Chime”, “Clarinet”, “Computer_keyboard”, “Cough”, “Cowbell”, “Double_bass”, “Drawer_open_or_close”, “Electric_piano”, “Fart”, “Finger_snapping”, “Fireworks”, “Flute”, “Glockenspiel”, “Gong”, “Gunshot_or_gunfire”, “Harmonica”, “Hi-hat”, “Keys_jangling”, “Knock”, “Laughter”, “Meow”, “Microwave_oven”, “Oboe”, “Saxophone”, “Scissors”, “Shatter”, “Snare_drum”, “Squeak”, “Tambourine”, “Tearing”, “Telephone”, “Trumpet”, “Violin_or_fiddle”, “Writing”

Here are some other relevant characteristics of FSDKaggle2018:

- The dataset is split into a train set and a test set.
- The train set is meant to be for system development and includes ~9.5k samples unequally distributed among 41 categories. The minimum number of audio samples per category in the train set is 94, and the maximum 300. The duration of the audio samples ranges from 300ms to 30s due to the diversity of the sound categories and the preferences of Freesound users when recording sounds.
- MFCC algorithm that I use will normalize the duration of inputs. So I do not need padding the data.
- Out of the ~9.5k samples from the train set, ~3.7k have manually-verified ground truth annotations and ~5.8k have non-verified annotations. The non-verified annotations of the train set have a quality estimate of at least 65-70% in each category. Checkout the Data labeling process section below for more information about this aspect.
- Non-verified annotations in the train set are properly flagged in train.csv so that participants can opt to use this information during the development of their systems.
- The test set is composed of ~1.6k samples with manually-verified annotations and with a similar category distribution than that of the train set. The test set is complemented with ~7.8k padding sounds which are not used for scoring the systems.

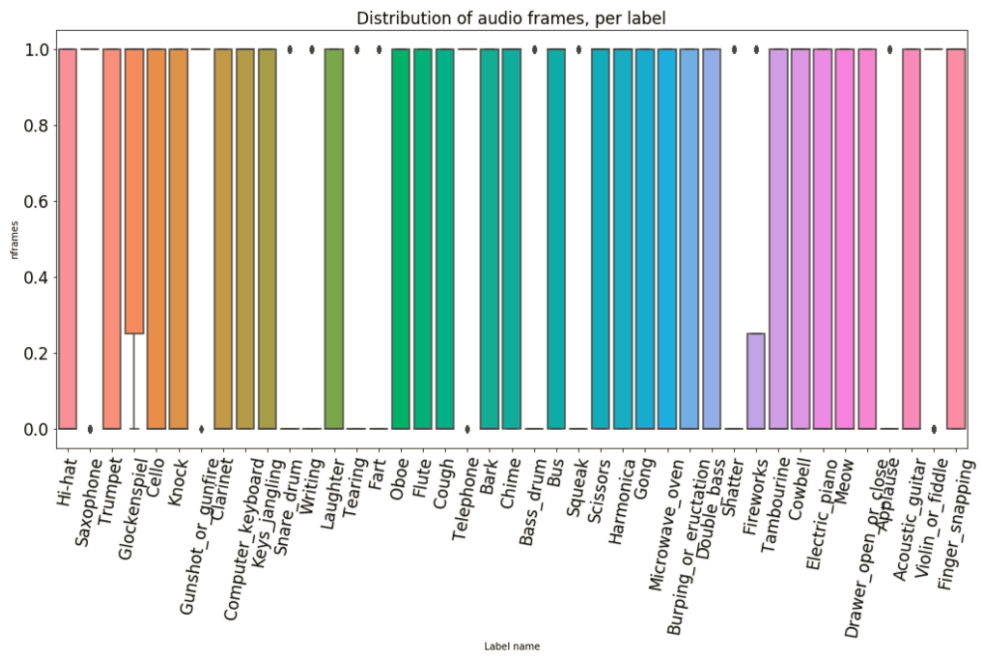






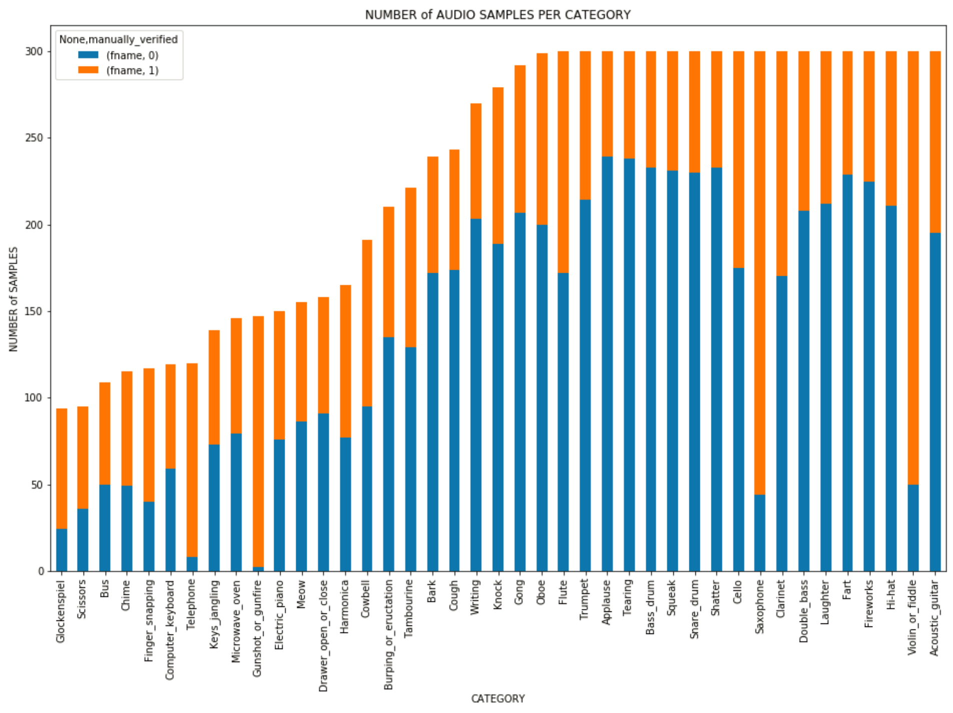
All audio samples in this dataset have a single label (i.e. are only annotated with one label). Checkout the Data labeling process section below for more information about this aspect.

Exploratory Visualization



Distribution of classes of train data set. Number of audio samples per category. This chart shows me the trauma of my train classes.

These graphs show me the different dimensions of train wav files. Some of the audio samples do not have a longer duration (frame differences).



Algorithms and Techniques

I'm using Mel Frequency Cepstral Coefficients (MFCC) algorithm to extract some usefull features from sound signals. Because of my hardware limits I cannot process whole raw signal files. So I need dimension reduction. MFCC algorithm helps me to do that. It took the raw data file. Does some maths. And give me a small input set. For more detail about the MFCC algorithm, see the Algorithm and Implementation.

In sound processing, the mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear transform of a log power spectrum on a nonlinear mel scale of frequency.

Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response. The linear-spaced frequency bands are used in the normal cepstrum. This frequency warping allows for a better representation of sound

MFCCs are commonly derived as follows:

1. Take the Fourier transform of (a windowed excerpt of) a signal.

2. Map the powers of the spectrum obtained above to the mel scale, using triangular overlapping windows.

3. Take the logs of the powers at each of the mel frequencies.

4. Take the discrete cosine transform of the list of mel log powers, as if it was a signal.

5. The MFCCs are the resulting spectrum.

Delta-delta "(first- and second-order frame-delta-delta) features such as" to-frame difference coefficients.

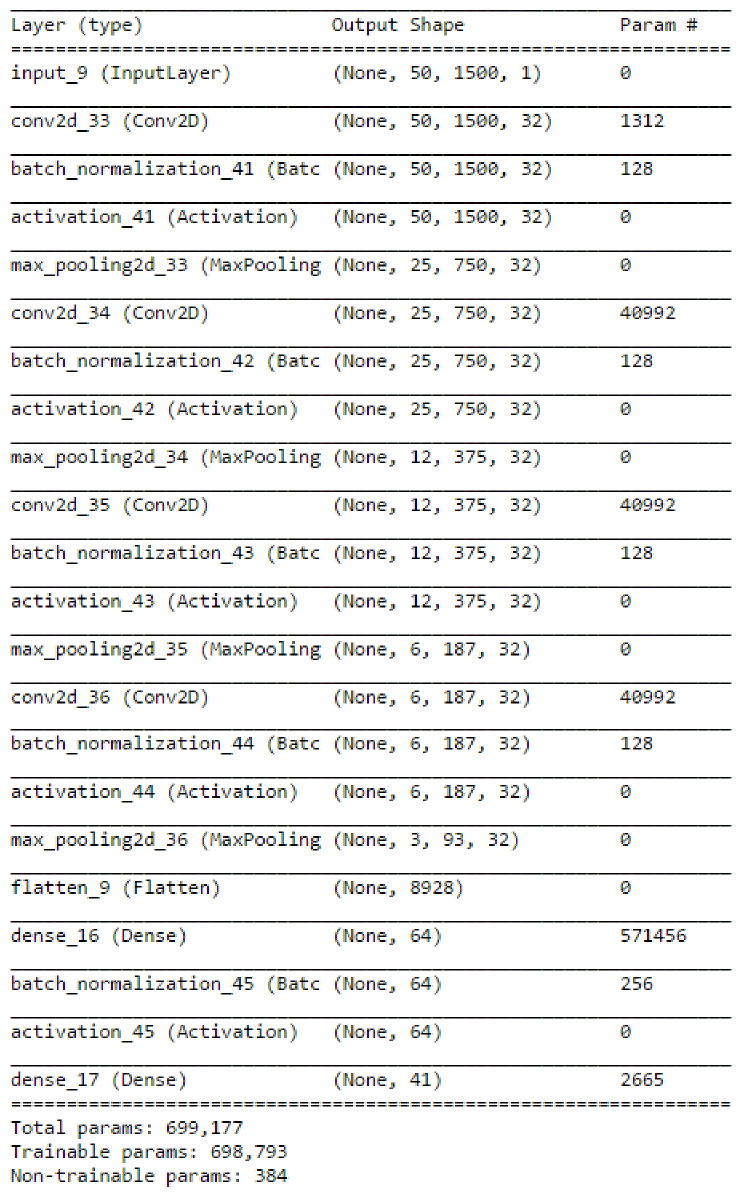
After feature extraction, I use a CNN model to train these features and learning effects of each feature. A Convolutional Neural Network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. CNNs use relatively little pre-processing image classification algorithms. This means that the network learns

The filters were hand-engineered in traditional algorithms. This independence from prior knowledge and human effort is a major advantage.

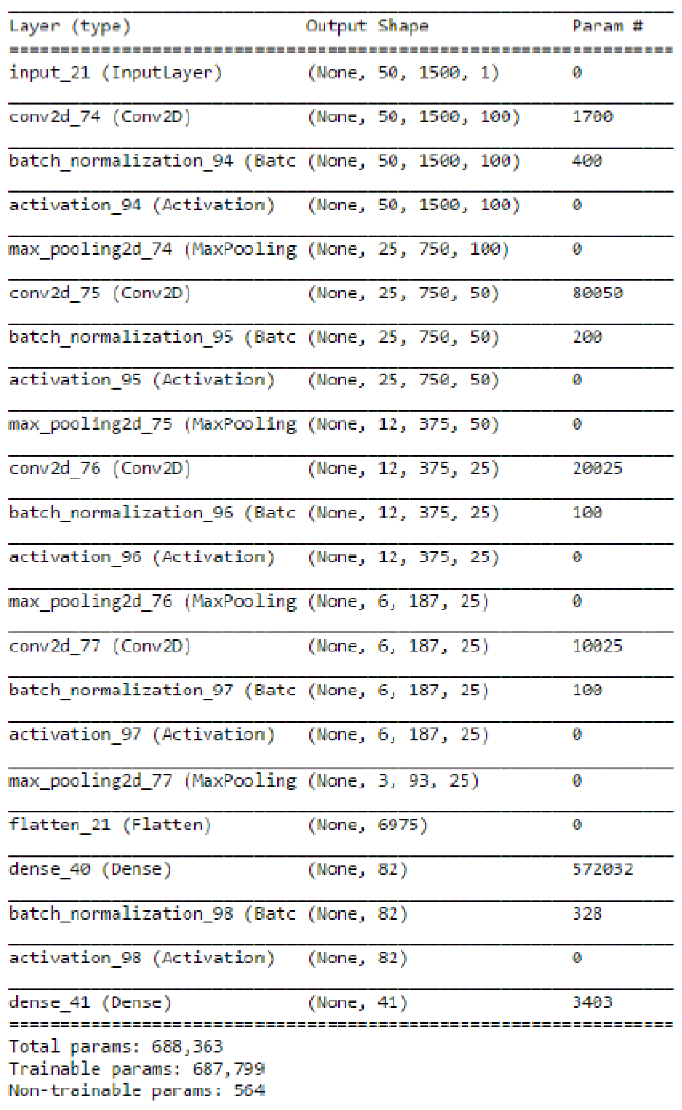
Benchmark

I have defined two models for comparison. They are guessing "a specially designed CNN Model" and a Random Choice.

Custom Designed CNN Model That Best Score



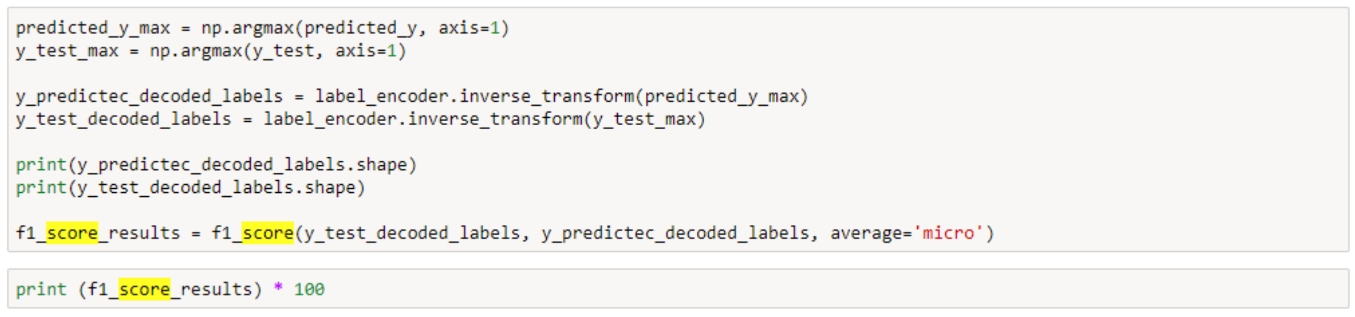
```
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         (None, 50, 1500, 1)       0
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 50, 1500, 32)      1312
_________________________________________________________________
batch_normalization_41 (Batc (None, 50, 1500, 32)      128
_________________________________________________________________
activation_41 (Activation)   (None, 50, 1500, 32)      0
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 25, 750, 32)       0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 25, 750, 32)       40992
_________________________________________________________________
batch_normalization_42 (Batc (None, 25, 750, 32)       128
_________________________________________________________________
activation_42 (Activation)   (None, 25, 750, 32)       0
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 12, 375, 32)       0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 12, 375, 32)       40992
_________________________________________________________________
batch_normalization_43 (Batc (None, 12, 375, 32)       128
_________________________________________________________________
activation_43 (Activation)   (None, 12, 375, 32)       0
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 6, 187, 32)        0
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 6, 187, 32)        40992
_________________________________________________________________
batch_normalization_44 (Batc (None, 6, 187, 32)        128
_________________________________________________________________
activation_44 (Activation)   (None, 6, 187, 32)        0
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 3, 93, 32)         0
_________________________________________________________________
flatten_9 (Flatten)          (None, 8928)              0
_________________________________________________________________
dense_16 (Dense)             (None, 64)                571456
_________________________________________________________________
batch_normalization_45 (Batc (None, 64)                256
_________________________________________________________________
activation_45 (Activation)   (None, 64)                0
_________________________________________________________________
dense_17 (Dense)             (None, 41)                2665
=================================================================
Total params: 699,177
Trainable params: 698,793
Non-trainable params: 384
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_21 (InputLayer)        (None, 50, 1500, 1)       0
_________________________________________________________________
conv2d_74 (Conv2D)           (None, 50, 1500, 100)     1700
_________________________________________________________________
batch_normalization_94 (Batc (None, 50, 1500, 100)     400
_________________________________________________________________
activation_94 (Activation)   (None, 50, 1500, 100)     0
_________________________________________________________________
max_pooling2d_74 (MaxPooling (None, 25, 750, 100)      0
_________________________________________________________________
conv2d_75 (Conv2D)           (None, 25, 750, 50)       80050
_________________________________________________________________
batch_normalization_95 (Batc (None, 25, 750, 50)       200
_________________________________________________________________
activation_95 (Activation)   (None, 25, 750, 50)       0
_________________________________________________________________
max_pooling2d_75 (MaxPooling (None, 12, 375, 50)       0
_________________________________________________________________
conv2d_76 (Conv2D)           (None, 12, 375, 25)       20025
_________________________________________________________________
batch_normalization_96 (Batc (None, 12, 375, 25)       100
_________________________________________________________________
activation_96 (Activation)   (None, 12, 375, 25)       0
_________________________________________________________________
max_pooling2d_76 (MaxPooling (None, 6, 187, 25)        0
_________________________________________________________________
conv2d_77 (Conv2D)           (None, 6, 187, 25)        10025
_________________________________________________________________
batch_normalization_97 (Batc (None, 6, 187, 25)        100
_________________________________________________________________
activation_97 (Activation)   (None, 6, 187, 25)        0
_________________________________________________________________
max_pooling2d_77 (MaxPooling (None, 3, 93, 25)         0
_________________________________________________________________
flatten_21 (Flatten)         (None, 6975)              0
_________________________________________________________________
dense_40 (Dense)             (None, 82)                572032
_________________________________________________________________
batch_normalization_98 (Batc (None, 82)                328
_________________________________________________________________
activation_98 (Activation)   (None, 82)                0
_________________________________________________________________
dense_41 (Dense)             (None, 41)                3403
=================================================================
Total params: 688,363
Trainable params: 687,799
Non-trainable params: 564
```

Another CNN Model (there are some kinds of parameters)

Random Selection estimates. And get 3.125 f1 points accuracy.



```
predicted_y_max = np.argmax(predicted_y, axis=1)
y_test_max = np.argmax(y_test, axis=1)

y_predictec_decoded_labels = label_encoder.inverse_transform(predicted_y_max)
y_test_decoded_labels = label_encoder.inverse_transform(y_test_max)

print(y_predictec_decoded_labels.shape)
print(y_test_decoded_labels.shape)

f1_score_results = f1_score(y_test_decoded_labels, y_predictec_decoded_labels, average='micro')
```

```
print (f1_score_results) * 100
```

Finally, I tested the specially designed CNN Model (the best tuned version) which gave me better accuracy. F1 score = 68%

III. Methodology

Data Preprocessing

Data preprocessing is a technique of data mining that involves an understandable transform of raw data. Real world data is often incomplete, inconsistent, and / or incomplete in certain behaviors or trends, and likely to contain many errors. Data preprocessing is a proven method for resolving such problems.

I have a large raw data file in my data cache that is difficult to process. And various data sizes due to different sound durations. So I am pre-processing the data and I have extracted some features so that the data size is reduced and I am giving the data to correct the size (data length) that I can process the data.

I have extracted some features using the MFCC algorithm. The anchor basically divides into fragments and summarizes the data framed. For more details, see Algorithm, Application. I filled out the MFCC results (signal data) because of different durations in my raw files. After the feature extraction and filling, it was seen that each signal file was 2D (50,1500)

Implementation

First, I read my raw files (.wav) and extract the mfcc properties by dividing the audio signal into 50 parts (n_mfcc = 50). I am saving the features extracted after extraction from the diskette as .mfcc files. (such as saving the CNN model). And the algorithm gives me 2d sequences (50,?). the first dimension has a length of 50, because I am using the n_mfcc parameter = 50 = 50

speak division in the audio signal division in the audio signal. And the second dimension has changed because of the duration of the different music signals. Which was a maximum of 1312. And to fill the input size for the CNN model, I filled all the signals up to 1500 sizes. After filling, I have a 2d sized matrix at the end (50,1500).

After the size reduction, I created a CNN model to find out how much influence the defining voice classification has. It basically combines the weights of each feature and features and weights to be learned by the CNN model. More details can be found here. I get my 3d CNN architecture (50.100.1) 3d dimensional input. And 41 binary classification outputs. I needed to binary encode my target label classes for binary classification. So I have a binary encoded output matrix that offers 41 different classes.

After trending and testing the model, I reverse the predicted ethics of a single hot encoding. And I compared the predicted F1 score with the label given using accuracy.

Refinement

1. I was trained and tested using less data in my initial training. (1500 total sounds for the train and 500 for the test, speed = 0.25). The accuracy of the CNN model was 16%.

Full data accuracy has increased to 33% and I continue to set other parameters.

2. I changed batch_size = 100 and epochs = 20 and the activation function of the last layer = 'sigmoid' (softmax) was increased to 47%

3. I was trained using the Batch_size parameter and was educated by using the 200, epochs parameter 6, decreasing epoch and increasing batch_size due to hardware and time constraints. I've updated the weights at a time for every 200 samples. And I've trained 6 times. These options give me enough accuracy right now. It also allows me to get results faster.

IV. Results

Model Evaluation and Validation

I tested some core layers and tested the results (f1 score). First, I tried to find filters (like RGB filter in image processing) so that I could apply audio signals, so I used the fold layers. I used the bulk normalization layer to normalize the output of layers for each layer. I am trying to summarize the data (selecting the maximum values) using the MaxPool2D layer. And at the end of my architecture I've copied this layer combination several times, adding the same combination, and trying to teach the correlation between different perceptions.

The architecture I mentioned above gives me the best score. I set some parameters (activation functions, kernel dimensions and filter parameters) and tested the results of the changing parameters.

I tried to enlarge it, but it caused an overheating problem. I did not use any layer like Dropout to handle overfitting.

The latest model Stratify Shuffle achieves 67% accuracy after changing the data splitting function to split. This partitioning method helped to further test the tradeoff and dataset classes in balance. I am still working on improving the accuracy of my CNN model.

I tested all my results using the F1 score.

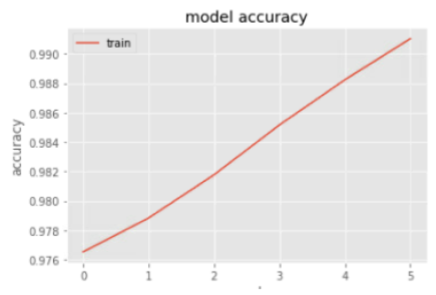
Justification

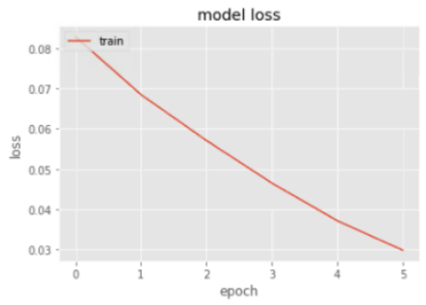
My benchmarking model is Random Selection I tagged all train data at random. And I see accuracy 2%

CNN model accuracy is much higher than the random selection option. 67%.

V. Conclusion

Free-Form Visualization





Below, as the epoch increases, the accuracy score increases. And model loss is reduced. But if there are too many episodes on the model, there will be problems of extreme distress for this model. Because I do not yet have over compression problems. Below you can see the accuracy and loss graphics of my architecture, but you can see 5 periods, I use transfer learning techniques.

Reflection

There is an analog signal raw yield while doing begging. I extracted some features using the MFCC algorithm. First, all my signals were divided into 50 chars and I got the second dimension, so I had 50 entries in the CNN model. But the truth was not okay. So I changed it (50.100) 2d dimension. I feed my CNN model with more detail.

The second critic's decision in my work. The output layer may be 1 node with scaling from 1 to 41, but I think it would be more accurate to calculate the classifications separately using 41 nodes in the last layer of CNN.

And of course, it's another tough thing to decide on CNN's architecture. I tried many models. Different layers and parameters. The best score is 67%. I'm still trying to improve my model.

Improvement

- The CNN model can be tested with different layers and parameters. Different studies can be done for this.
- There are many other audio features that can be applied to CNN. We can apply them.
- Fully tiered layers will develop the model but will require more resources. Data enlargement can help. This requires data and more.
- Dropout layers, L* regulations and incremental data can also be useful to avoid problems of excessive asylum.
- Some data scientists with knowledge of area knowledge can help or share pre-compiled models. Some pre-built models will help. Thus, it can create a mode in pre-computed models.