SIT378 Speech Classification

This study is for classifying speech data. In order to classify audio data, it is divided into two parts. Part 1 is the preprocessing of audio data. Part 2 actually classifies using machine learning and other classification techniques.

Part 1

1. Loading data

* **Installing the library**  
    
   We use a library called librosa to handle audio data. Run the following on Linux or Windows with Python installed:

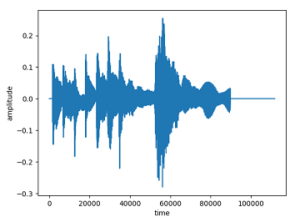
|  |
| --- |
| pip install librasa |

* **Loading audio data**  
    
   Just as you can read an image as an array of numbers, you can do the same with audio. Readable file formats are wav, mp3.

|  |
| --- |
| Import librasa  y, sr= librasa.load(‘file name’)  y= n umpy.arrray with continuous amplitude values containing audio data  sr = Sample ring rate value, number of audio data after 1 second. |

1. View the waveform of the audio data
   * **Painting the waveform of the audio data The**  
       
      audio data called by librosa.load can draw a graph of Sin waves.

|  |
| --- |
| 1. import librosa 2. import matplotlib.pyplot as plt 3. y, sr = librosa.load('file name') 4. plt.plot(y) 5. plt.show() |



1. **Augmentation**

Like image classification, audio data is augmented and inflated.

This time, white noise, shift, and stretch are added to the voice data.

The following is wave data with augmentation based on dog barking.

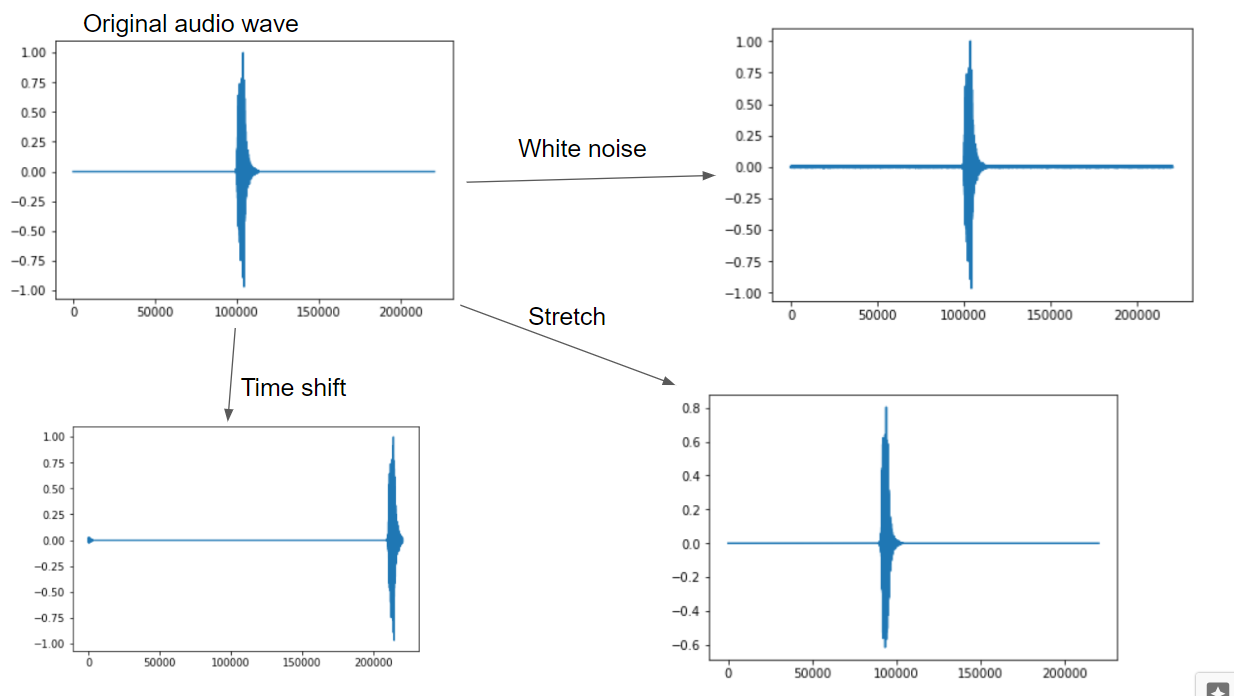
Data augmentation is the process of creating new synthetic training samples by adding small perturbations to the initial training set.

The aim is to make the model immutable to these perturbations and to enhance its ability to generalize.

For this to work, the perturbation addition must save the same label as the original training sample.

Images can be expanded by shifting, zooming, rotating, and so on.

In our case, add noise, stretch and roll, pitch shift.



|  |
| --- |
| 1.  # data augmentation: add white noise  2. def add\_white\_noise(y, rate=0.002):  3. return x + rate\*np.random.randn(len(y))  4.  5. # data augmentation: shift sound in timeframe  6. def shift\_sound(y, rate=2):  7. return np.roll(y, int(len(y)//rate))  8.  9. # data augmentation: stretch sound  10. def stretch\_sound(y, rate=1.1):  11. input\_length = len(y)  12. y = librosa.effects.time\_stretch(y, rate)  13. if len(y)>input\_length:  14. return y[:input\_length]  15. else:  16. return np.pad(y, (0, max(0, input\_length - len(y))), "constant") |

1. **Calculate the average value of the amplitude**

The average amplitude value of each audio data is calculated and compared with other data to check whether the sound has characteristics. If you have a lot of audio data, you can use it for machine learning data.

|  |
| --- |
| * + **import librosa**   + **import numpy as np**   + **y, sr = librosa.load('onsei.wav')**   + **y\_mean = np.sqrt(np.mean(y=y\*\*2,axis=1))** |

1. **Calculate the number of zero-crossers**

The zero cross number is a numerical value that indicates the degree of noise, and it is calculated for each voice data to see if there are any features compared to other data. If you have a lot of audio data, you can use it as machine learning data.

|  |
| --- |
| * **import librosa** * **y, sr = librosa.load('onsei.wav')** * **zc = librosa.zero\_crossings(y)** |

1. **Calculate the Mel Spectrogram**

The audio waveform is decomposed into multiple SIN waves, and information on frequency, amplitude, and time is provided, and the data that takes into account roughness in high-pitched bass is called a mel spectrogram. Visualize to see features and use it as machine learning data.

|  |
| --- |
| import librosa import matplotlib.pyplot as plt  y, sr = librosa.load('onsei.wav')  melspec = librosa.feature.melspectrogram(y, sr)  # Conversion of decibel to human-understandable volume  melspec\_db = librosa.amplitude\_to\_db(melspec)  # Visualization  librosa.display.specshow(melspec\_db)  plt.show() |

Part 2

I think there are various classification methods in this part, but this time we will classify using machine learning.

This time, we calculated features such as "average amplitude value", "zero cross number", and "mel spectrogram" for voice data. These can be machine learning data as inputs (explanatory variables).

|  |
| --- |
| import glob  import librosa  import numpy as np  import matplotlib.pyplot as plt  from sklearn import preprocessing    # Read multiple data  dataset = []  melspecs = []  for file\_name in glob.glob('dir/\*.wav'):  y, sr = librosa.load('onsei.wav')  dataset.append(y)    # Mel Spectrogram Calculation  melspec = librosa.feature.melspectrogram(y, sr)  melspec = librosa.amplitude\_to\_db(melspec).flatten()  melspecs.append(melspec.astype(np.float16))    # Average amplitude of each data  mean = np.sqrt(np.mean(dataset\*\*2,axis=1))    # Number of zero crosses for each data  zc = np.sum(librosa.zero\_crossings(dataset),axis=1)    # Use machine train\_feature training data  train\_feature = pd. DataFrame()  train\_feature['mean'] = mean  train\_feature['zc'] = zc  train\_feature['melspecs'] = melspecs  labels = #Include labels representing classifications in audio data    # If there is a feature in the above feature, use the supervised learning algorithm,  # If not, you can classify it as unsupervised learning.  # Some algorithms require standardization of numerical values.    # Machine learning: The following example uses supervised RandomForest.  from sklearn.ensemble import RandomForestClassifier  model = RandomForestClassifier()  model.fit(train\_feature, train\_labels)    # Prediction  pred = model.predict(test\_feature) |

I will also study how to classify speech with deep learning.

In this study, we will use the dataset ESC-50.

The classification method used is CNN (Convolutional Neural Network).

Environmental sounds and natural sounds are classified by the Convolutional neural network. The target is voiceless sounds (words) such as animal sounds, rain sounds, human coughs, clock alarms, and engine sounds  
.

Using these sounds, I made a classifier by following the steps below.

1. Pre-treatment of voice data
   1. Data input
   2. Augmentation
   3. Melspectrogrum
   4. Data usage
2. Classified on CNN
   1. CNN Principles
   2. Using Amsgrad as an optimization function
   3. Using mixup for training data

## Pre-treatment of voice data

[ESC-50 is a dataset of](https://github.com/karoldvl/ESC-50) 2,000 files of 50 classes of environmental sounds.   
The classes include:

A screenshot of a computer

Description automatically generated with low confidence

There are 50 classes, and 40 files of data for each class are prepared, for a total of 2,000 files.   
One file is 5 seconds long and has a sampling rate of 44,100.

**Data input**

You can download the data below.   
<https://github.com/karoldvl/ESC-50>

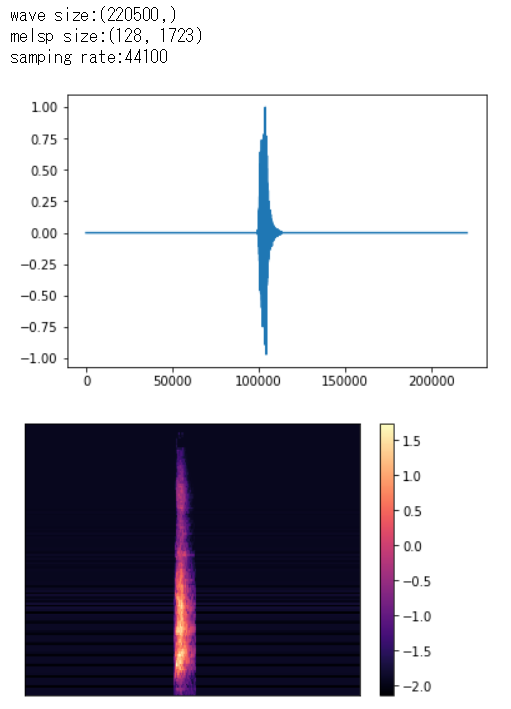
The contents are audio data (.wav) and metadata (.csv). The metadata contains the file name, class (  
0-49), and class name (see table above).

Audio data (.wav) [can be handled as](https://librosa.github.io/librosa/) numpy.array by loading it with a library called librosa.

|  |
| --- |
| **import os**  **import random**  **import numpy as np**  **import pandas as pd**  **import librosa**  **import librosa.display**  **import matplotlib.pyplot as plt**  **import seaborn as sn**  **from sklearn import model\_selection**  **from sklearn import preprocessing**  **import IPython.display as ipd**  **# define directories**  **base\_dir = "./"**  **esc\_dir = os.path.join(base\_dir, "ESC-50-master")**  **meta\_file = os.path.join(esc\_dir, "meta/esc50.csv")**  **audio\_dir = os.path.join(esc\_dir, "audio/")**  **# load metadata**  **meta\_data = pd.read\_csv(meta\_file)**  **# get data size**  **data\_size = meta\_data.shape**  **print(data\_size)**  **# arrange target label and its name**  **class\_dict = {}**  **for i in range(data\_size[0]):**  **if meta\_data.loc[i,"target"] not in class\_dict.keys():**  **class\_dict[meta\_data.loc[i,"target"]] = meta\_data.loc[i,"category"]** |

The next step is to load the WAV data and draw the waveform data and mel spectrogram.

|  |
| --- |
| **# load a wave data**  **def load\_wave\_data(audio\_dir, file\_name):**  **file\_path = os.path.join(audio\_dir, file\_name)**  **x, fs = librosa.load(file\_path, sr=44100)**  **return x,fs**  **# change wave data to mel-stft**  **def calculate\_melsp(x, n\_fft=1024, hop\_length=128):**  **stft = np.abs(librosa.stft(x, n\_fft=n\_fft, hop\_length=hop\_length))\*\*2**  **log\_stft = librosa.power\_to\_db(stft)**  **melsp = librosa.feature.melspectrogram(S=log\_stft,n\_mels=128)**  **return melsp**  **# display wave in plots**  **def show\_wave(x):**  **plt.plot(x)**  **plt.show()**  **# display wave in heatmap**  **def show\_melsp(melsp, fs):**  **librosa.display.specshow(melsp, sr=fs)**  **plt.colorbar()**  **plt.show()**  **# example data**  **x, fs = load\_wave\_data(audio\_dir, meta\_data.loc[0,"filename"])**  **melsp = calculate\_melsp(x)**  **print("wave size:{0}\nmelsp size:{1}\nsamping rate:{2}".format(x.shape, melsp.shape, fs))**  **show\_wave(x)**  **show\_melsp(melsp, fs)** |

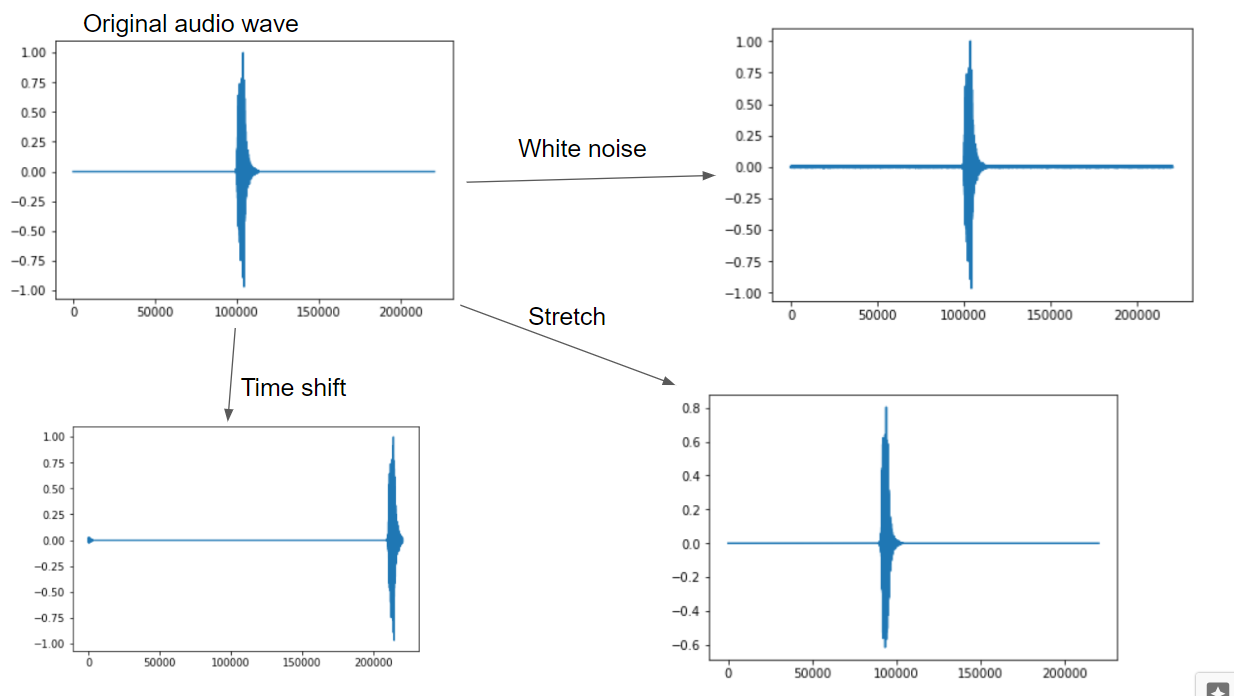


### Augmentation

Like image classification, audio data is augmented and inflated.   
The Augmentation method is used as a reference here.

This time, white noise, shift, and stretch are added to the voice data.

The following is wave data with augmentation based on dog barking.



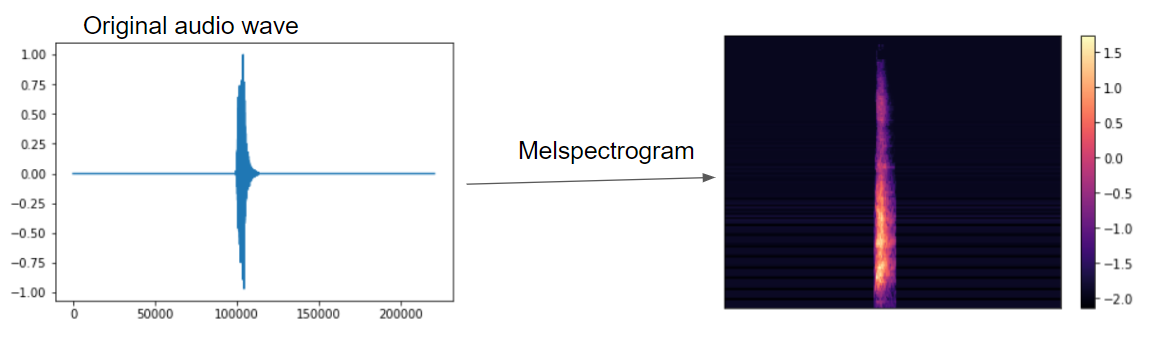
|  |
| --- |
| **# data augmentation: add white noise**  **def add\_white\_noise(x, rate=0.002):**  **return x + rate\*np.random.randn(len(x))**  **# data augmentation: shift sound in timeframe**  **def shift\_sound(x, rate=2):**  **return np.roll(x, int(len(x)//rate))**  **# data augmentation: stretch sound**  **def stretch\_sound(x, rate=1.1):**  **input\_length = len(x)**  **x = librosa.effects.time\_stretch(x, rate)**  **if len(x)>input\_length:**  **return x[:input\_length]**  **else:**  **return np.pad(x, (0, max(0, input\_length - len(x))), "constant")** |

### Mel Spectrogram

WAV data itself is one-dimensional time series data.   
This is applied by a filter bank of mel frequencies to obtain a mel spectrogram.   
This can be obtained in [librosa.feature.melspectrogram.](https://librosa.github.io/librosa/generated/librosa.feature.melspectrogram.html)

|  |
| --- |
| **import librosa**  **import librosa.display**  **# change wave data to mel-stft**  **def calculate\_melsp(x, n\_fft=1024, hop\_length=128):**  **stft = np.abs(librosa.stft(x, n\_fft=n\_fft, hop\_length=hop\_length))\*\*2**  **log\_stft = librosa.power\_to\_db(stft)**  **melsp = librosa.feature.melspectrogram(S=log\_stft,n\_mels=128)**  **return melsp** |

A picture containing diagram, screenshot, line, rectangle

Description automatically generated

### Preparing the data

So, the wav data is augmented into a mel spectrogram and is used as the input data of the classifier.   
As a preprocessing, prepare all wav files as Augmentation and Mel Spectrogram.

First, separate the training data and the test data.   
Test data should be 25%, and stratify=y will divide each class equally.

|  |
| --- |
| **# get training dataset and target dataset**  **x = list(meta\_data.loc[:,"filename"])**  **y = list(meta\_data.loc[:, "target"])**  **x\_train, x\_test, y\_train, y\_test = model\_selection.train\_test\_split(x, y, test\_size=0.25, stratify=y)**  **print("x train:{0}\ny train:{1}\nx test:{2}\ny test:{3}".format(len(x\_train),**  **len(y\_train),**  **len(x\_test),**  **len(y\_test)))**  **"""output**  **x train:1500**  **y train:1500**  **x test:500**  **y test:500**  **"""**  **# showing the classes are equally splitted**  **a = np.zeros(50)**  **for c in y\_test:**  **a[c] += 1**  **print(a)**  **"""output**  **[10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.**  **10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.**  **10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]**  **"""** |

Save both training data and test data as NPZ. The training data is further saved separately by white noise, shift, stretching, and a combination thereof. As a result, the following data  
  
 is prepared in NPZ format.

* Mel spectrogram of test data (500 points)
* Mel spectrogram of training data (1,500 points)
* Mel spectrogram of training data with white noise (1,500 points)
* Mel spectrogram of shifted training data (1,500 points)
* Mel spectrogram of stretched training data (1,500 points)
* Mel spectrogram of training data with random combination of white noise, shift, and stretching (1,500 points)

|  |
| --- |
| * **freq = 128** * **time = 1723** * **# save wave data in npz, with augmentation** * **def save\_np\_data(filename, x, y, aug=None, rates=None):** * **np\_data = np.zeros(freq\*time\*len(x)).reshape(len(x), freq, time)** * **np\_targets = np.zeros(len(y))** * **for i in range(len(y)):** * **\_x, fs = load\_wave\_data(audio\_dir, x[i])** * **if aug is not None:** * **\_x = aug(x=\_x, rate=rates[i])** * **\_x = calculate\_melsp(\_x)** * **np\_data[i] = \_x** * **np\_targets[i] = y[i]** * **np.savez(filename, x=np\_data, y=np\_targets)** * **# save test dataset** * **if not os.path.exists("esc\_melsp\_test.npz"):** * **save\_np\_data("esc\_melsp\_test.npz", x\_test, y\_test)** * **# save raw training dataset** * **if not os.path.exists("esc\_melsp\_train\_raw.npz"):** * **save\_np\_data("esc\_melsp\_train\_raw.npz", x\_train, y\_train)** * **# save training dataset with white noise** * **if not os.path.exists("esc\_melsp\_train\_wn.npz"):** * **rates = np.random.randint(1,50,len(x\_train))/10000** * **save\_np\_data("esc\_melsp\_train\_wn.npz", x\_train, y\_train, aug=add\_white\_noise, rates=rates)** * **# save training dataset with sound shift** * **if not os.path.exists("esc\_melsp\_train\_ss.npz"):** * **rates = np.random.choice(np.arange(2,6),len(y\_train))** * **save\_np\_data("esc\_melsp\_train\_ss.npz", x\_train, y\_train, aug=shift\_sound, rates=rates)** * **# save training dataset with stretch** * **if not os.path.exists("esc\_melsp\_train\_st.npz"):** * **rates = np.random.choice(np.arange(80,120),len(y\_train))/100** * **save\_np\_data("esc\_melsp\_train\_st.npz", x\_train, y\_train, aug=stretch\_sound, rates=rates)** * **# save training dataset with combination of white noise and shift or stretch** * **if not os.path.exists("esc\_melsp\_train\_com.npz"):** * **np\_data = np.zeros(freq\*time\*len(x\_train)).reshape(len(x\_train), freq, time)** * **np\_targets = np.zeros(len(y\_train))** * **for i in range(len(y\_train)):** * **x, fs = load\_wave\_data(audio\_dir, x\_train[i])** * **x = add\_white\_noise(x=x, rate=np.random.randint(1,50)/1000)** * **if np.random.choice((True,False)):** * **x = shift\_sound(x=x, rate=np.random.choice(np.arange(2,6)))** * **else:** * **x = stretch\_sound(x=x, rate=np.random.choice(np.arange(80,120))/100)** * **x = calculate\_melsp(x)** * **np\_data[i] = x** * **np\_targets[i] = y\_train[i]** * **np.savez("esc\_melsp\_train\_com.npz", x=np\_data, y=np\_targets)** |

As a result, 500 test data and 7,500 training data are prepared.

## Environmental sounds by deep learning

Classify environmental sounds with a Convolutional neural network.

### Definition of CNN

The neural network has the following structure:

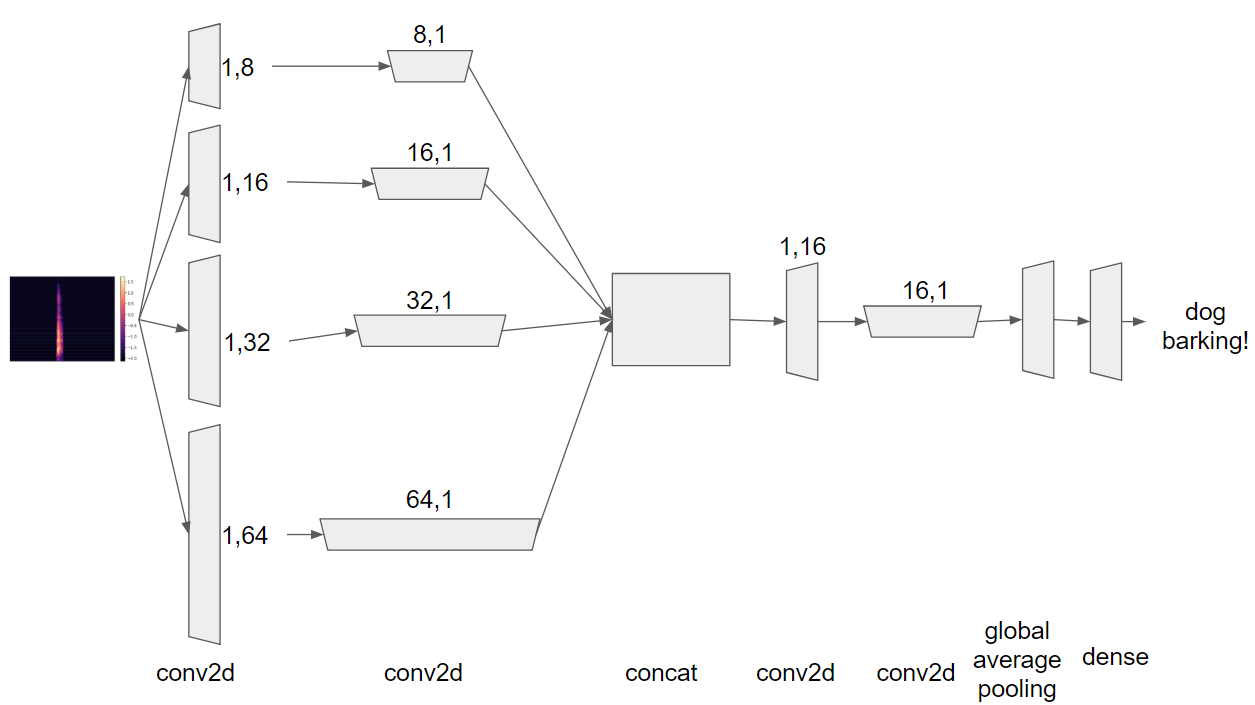
|  |
| --- |
| **import keras**  **from keras.models import Model**  **from keras.layers import Input, Dense, Dropout, Activation**  **from keras.layers import Conv2D, GlobalAveragePooling2D**  **from keras.layers import BatchNormalization, Add**  **from keras.callbacks import EarlyStopping, ModelCheckpoint**  **# redefine target data into one hot vector**  **classes = 50**  **y\_train = keras.utils.to\_categorical(y\_train, classes)**  **y\_test = keras.utils.to\_categorical(y\_test, classes)**  **def cba(inputs, filters, kernel\_size, strides):**  **x = Conv2D(filters, kernel\_size=kernel\_size, strides=strides, padding='same')(inputs)**  **x = BatchNormalization()(x)**  **x = Activation("relu")(x)**  **return x**  **# define CNN**  **inputs = Input(shape=(x\_train.shape[1:]))**  **x\_1 = cba(inputs, filters=32, kernel\_size=(1,8), strides=(1,2))**  **x\_1 = cba(x\_1, filters=32, kernel\_size=(8,1), strides=(2,1))**  **x\_1 = cba(x\_1, filters=64, kernel\_size=(1,8), strides=(1,2))**  **x\_1 = cba(x\_1, filters=64, kernel\_size=(8,1), strides=(2,1))**  **x\_2 = cba(inputs, filters=32, kernel\_size=(1,16), strides=(1,2))**  **x\_2 = cba(x\_2, filters=32, kernel\_size=(16,1), strides=(2,1))**  **x\_2 = cba(x\_2, filters=64, kernel\_size=(1,16), strides=(1,2))**  **x\_2 = cba(x\_2, filters=64, kernel\_size=(16,1), strides=(2,1))**  **x\_3 = cba(inputs, filters=32, kernel\_size=(1,32), strides=(1,2))**  **x\_3 = cba(x\_3, filters=32, kernel\_size=(32,1), strides=(2,1))**  **x\_3 = cba(x\_3, filters=64, kernel\_size=(1,32), strides=(1,2))**  **x\_3 = cba(x\_3, filters=64, kernel\_size=(32,1), strides=(2,1))**  **x\_4 = cba(inputs, filters=32, kernel\_size=(1,64), strides=(1,2))**  **x\_4 = cba(x\_4, filters=32, kernel\_size=(64,1), strides=(2,1))**  **x\_4 = cba(x\_4, filters=64, kernel\_size=(1,64), strides=(1,2))**  **x\_4 = cba(x\_4, filters=64, kernel\_size=(64,1), strides=(2,1))**  **x = Add()([x\_1, x\_2, x\_3, x\_4])**  **x = cba(x, filters=128, kernel\_size=(1,16), strides=(1,2))**  **x = cba(x, filters=128, kernel\_size=(16,1), strides=(2,1))**  **x = GlobalAveragePooling2D()(x)**  **x = Dense(classes)(x)**  **x = Activation("softmax")(x)**  **model = Model(inputs, x)**  **# initiate Adam optimizer**  **opt = keras.optimizers.adam(lr=0.00001, decay=1e-6, amsgrad=True)**  **# Let's train the model using Adam with amsgrad**  **model.compile(loss='categorical\_crossentropy',**  **optimizer=opt,**  **metrics=['accuracy'])**  **model.summary()** |

After convolving (cba) with multiple filters of different lengths, it is combined (Add) and further convolved, and GlobalAveragePooling is used to classify them.   
This is because filters with different lengths add length and shortness to the convolution of the sound range and the convolution of the time axis.   
For example, in the following, it is convolved with a filter of length 8 in the direction of the sound range and then with a filter of length 8 in the time axis direction.

|  |
| --- |
| **x\_1 = cba(inputs, filters=32, kernel\_size=(1,8), strides=(1,2))**  **x\_1 = cba(x\_1, filters=32, kernel\_size=(8,1), strides=(2,1))** |

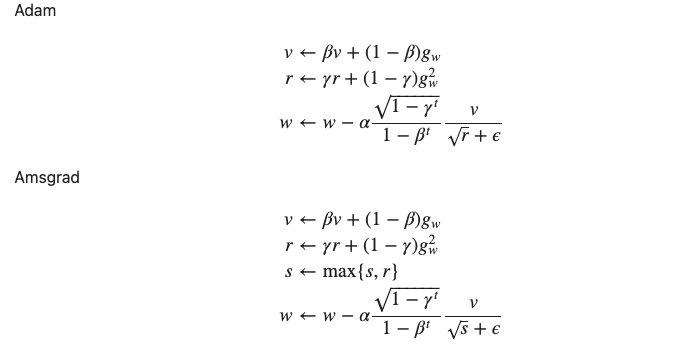
Filter lengths of 8, 16, 32, and 64 provide variety in the length to be convolved.

Use [GlobalAveragePooling for the taxonomic layer](https://qiita.com/mine820/items/1e49bca6d215ce88594a).   
GlobalAveragePooling aims to reduce the number of parameters.



**Using Amsgrad** as an optimization function

For optimization functions, add the amsgrad option to Adam.   
[Amsgrad](https://openreview.net/forum?id=ryQu7f-RZ) is an optimization [function published at the end](https://openreview.net/forum?id=ryQu7f-RZ) of [2017](https://openreview.net/forum?id=ryQu7f-RZ) and [is an](https://openreview.net/forum?id=ryQu7f-RZ) improved version of Adam. By reducing the learning rate sufficiently, it is expected to be   
better accuracy and prevent overfitting than Adam. Numerically, it is as follows  
.

****

**Using** **mixup** for training data

[Mixup is used for training data](https://arxiv.org/abs/1710.09412).   
Mixup mixes training data in two pairs. By doing this, it will be possible to more clearly separate the feature space between classes  
.

Mixup[1](https://qiita.com/yu4u/items/70aa007346ec73b7ff05#fn1) is one of the data augmentation methods that mixes pairs of two training samples to create a new training sample.   
Specifically, data and label pairs (X 1,y1), (X 2, From y 2), create a new training sample (X,y) by the following equation. Here, it is assumed that the labels y1 and y2 are vectors of the one-hot representation. X 1,X 2 is any vector or tensor. A picture containing font, text, white, calligraphy

Description automatically generated

Here, λ∈[0,1] is obtained by sampling from the beta distribution Be(α,α), and the α is a hyperparameter. It is characteristic not only the data X 1,X 2, but also the label y 1,y 2 is also a point that mixes.

|  |
| --- |
| **import numpy as np**  **class MixupGenerator():**  **def \_\_init\_\_(self, X\_train, y\_train, batch\_size=32, alpha=0.2, shuffle=True, datagen=None):**  **self. X\_train = X\_train**  **self.y\_train = y\_train**  **self.batch\_size = batch\_size**  **self.alpha = alpha**  **self.shuffle = shuffle**  **self.sample\_num = len(X\_train)**  **self.datagen = datagen**  **def \_\_call\_\_(self):**  **while True:**  **indexes = self.\_\_get\_exploration\_order()**  **itr\_num = int(len(indexes) // (self.batch\_size \* 2))**  **for i in range(itr\_num):**  **batch\_ids = indexes[i \* self.batch\_size \* 2:(i + 1) \* self.batch\_size \* 2]**  **X, y = self.\_\_data\_generation(batch\_ids)**  **yield X, y**  **def \_\_get\_exploration\_order(self):**  **indexes = np.arange(self.sample\_num)**  **if self.shuffle:**  **np.random.shuffle(indexes)**  **return indexes**  **def \_\_data\_generation(self, batch\_ids):**  **\_, h, w, c = self. X\_train.shape**  **\_, class\_num = self.y\_train.shape**  **X1 = self. X\_train[batch\_ids[:self.batch\_size]]**  **X2 = self. X\_train[batch\_ids[self.batch\_size:]]**  **y1 = self.y\_train[batch\_ids[:self.batch\_size]]**  **y2 = self.y\_train[batch\_ids[self.batch\_size:]]**  **l = np.random.beta(self.alpha, self.alpha, self.batch\_size)**  **X\_l = l.reshape(self.batch\_size, 1, 1, 1)**  **y\_l = l.reshape(self.batch\_size, 1)**  **X = X1 \* X\_l + X2 \* (1 - X\_l)**  **y = y1 \* y\_l + y2 \* (1 - y\_l)**  **if self.datagen:**  **for i in range(self.batch\_size):**  **X[i] = self.datagen.random\_transform(X[i])**  **return X, y** |

**URL list**

[**https://github.com/karolpiczak/ESC-50**](https://github.com/karolpiczak/ESC-50)

[**https://librosa.org/doc/latest/index.html**](https://librosa.org/doc/latest/index.html)

[**https://www.kaggle.com/code/CVxTz/audio-data-augmentation/notebook**](https://www.kaggle.com/code/CVxTz/audio-data-augmentation/notebook)

[**https://arxiv.org/pdf/1312.4400.pdf**](https://arxiv.org/pdf/1312.4400.pdf)

[**https://openreview.net/forum?id=ryQu7f-RZ**](https://openreview.net/forum?id=ryQu7f-RZ)

[**https://fdlm.github.io/post/amsgrad/**](https://fdlm.github.io/post/amsgrad/)

[**https://arxiv.org/abs/1710.09412**](https://arxiv.org/abs/1710.09412)

[**https://openreview.net/forum?id=B1Gi6LeRZ**](https://openreview.net/forum?id=B1Gi6LeRZ)

[**https://www.inference.vc/mixup-data-dependent-data-augmentation/**](https://www.inference.vc/mixup-data-dependent-data-augmentation/)