Foundations of Deep Learning
Master’s Degree in Data Science
Academic Year 2021/2022

TUT Acoustic Scenes 201

Group members:

Michele Salvaterra (891109)
Luca Sammarini (884591)
Eugenio Tarolli Bramè (889038)

Aim

The aim of the project is to **classify correctly** 4680 audio files containing ambient sound registered from 15 different acoustic scenes.
Acoustic scene classification is a fundamental task for environmental awareness used for example for devices, robots etc.

Dataset description

- Collected in Finland by Tampere University of Technology between June 2015 and January 2016.

- Contains 4680 audio tracks lasting 10 seconds each, taken from a set of 15 scenes, having a size of 10.7 GB. All the audio tracks were captured with a frequence of 44100 Hz and a resolution of 24 bits.

- Completely balanced: each acoustic scene has 312 segments totaling 52 minutes of audio. For all acoustic scenes, 3-5 minutes long recordings were captured each in a different location.

Dataset description

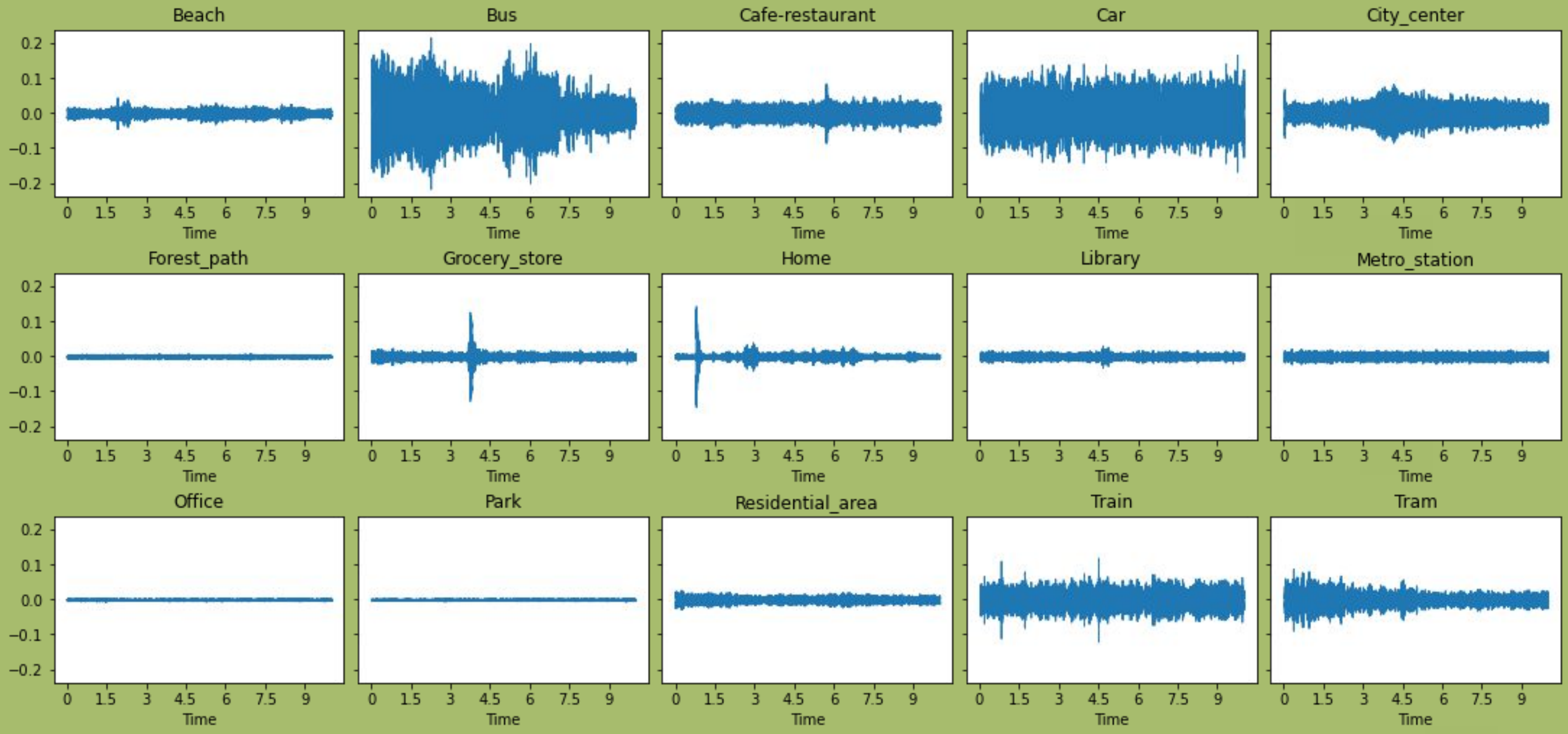
Instances: 4680

Classes: 15

Instances for each class: 312
(Completely balanced)

Class	Instances
Beach	312
Bus	312
Cafe-Restaurant	312
Car	312
City-center	312
Forest-path	312
Grocery-store	312
Home	312
Library	312
Metro-station	312
Office	312
Park	312
Residential-area	312
Train	312
Tram	312

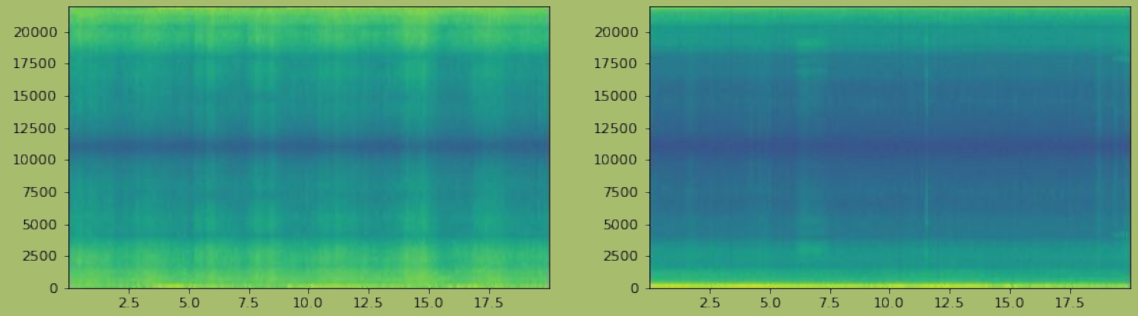
Audio examples



Audio preprocessing

1) Audio clips were converted from 24 bits audio to 16 bits audio to simplify the preprocessing phase.
2) Each audio clip was transformed into a spectrogram.

We initially used a library called ***wave*** to generate the spectrograms, but it was not so efficient, so we decided to create new spectrograms using **librosa**. We generated two kinds of spectrograms: STFTs and MFCCs.



Spectrograms of samples from Beach and Bus classes respectively, generated with the *wave* library.

STFT v.s. MFCC

STFT
(Short Time Fourier Transform)

Is a Fourier-related procedure used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time.

The procedure is executed dividing the signal in small windows and computing the FT on each window. This leads to the spectrum for every segment. Finally, the spectrum is plotted as a function of time.

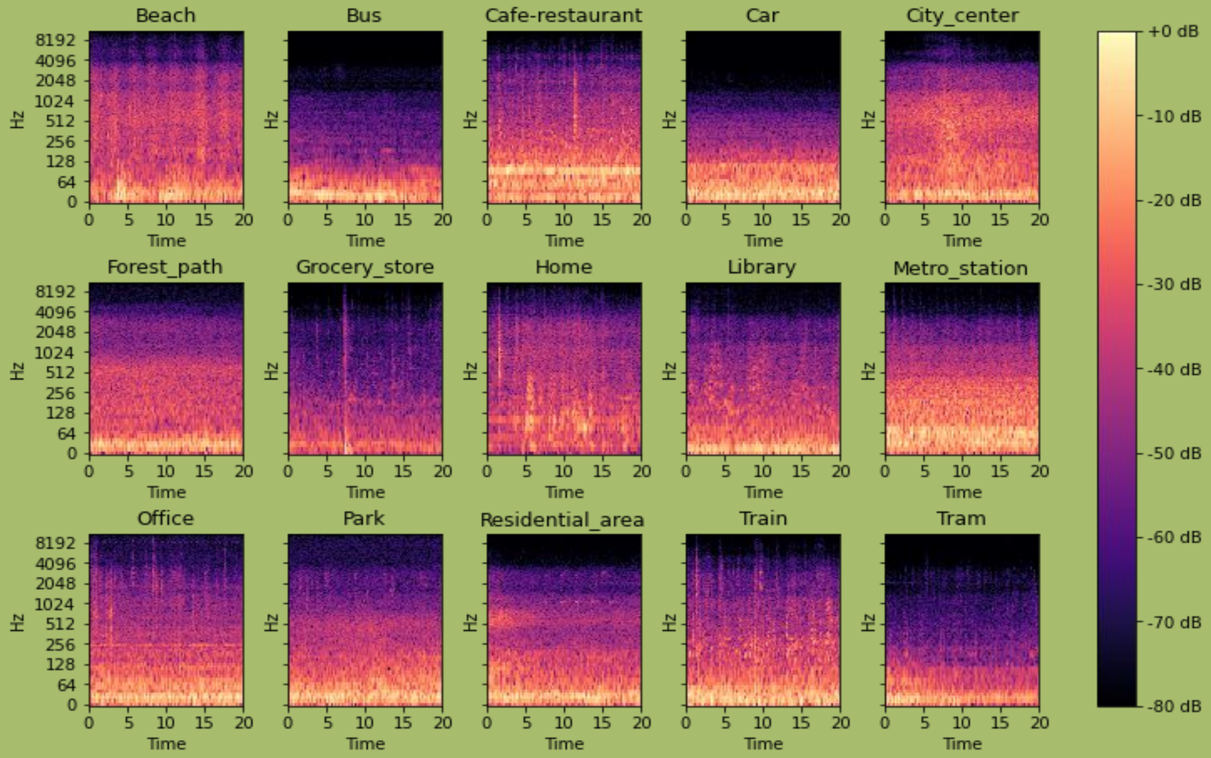
MFCCs
(Mel Frequency Cepstral Coefficients)

Are coefficients that collectively make up an MFC.

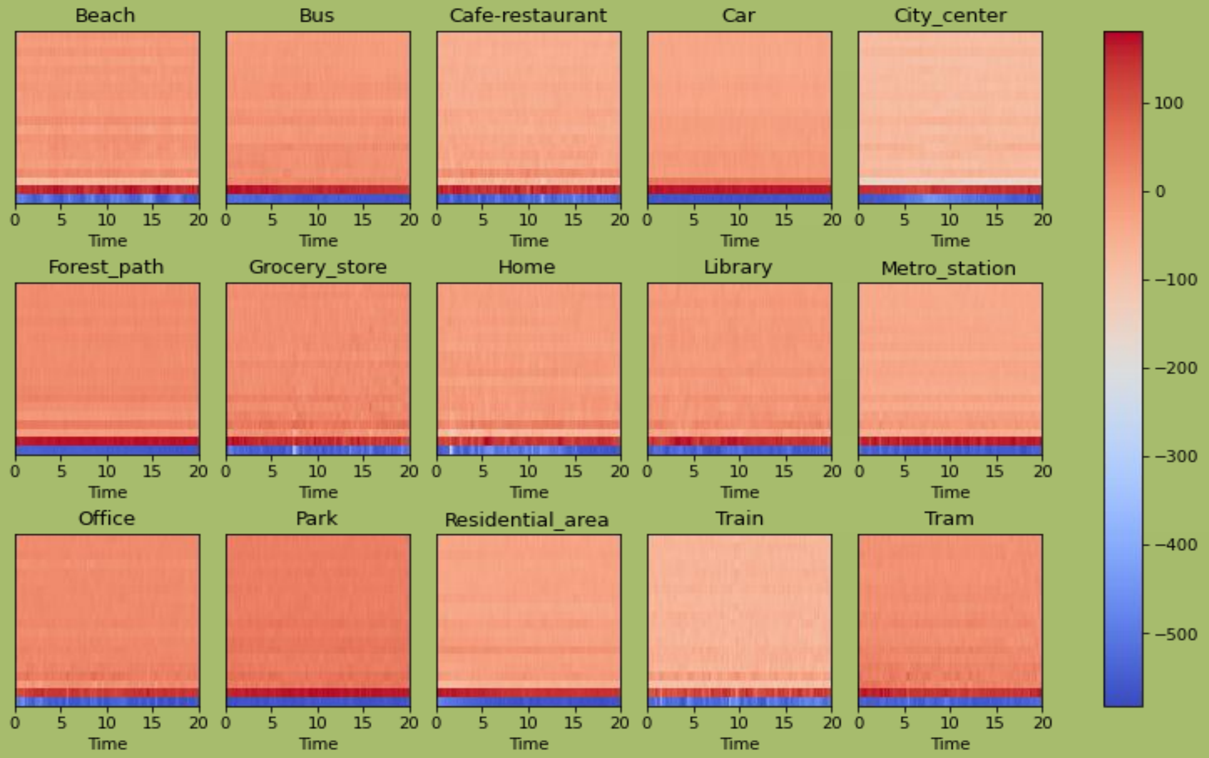
Mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

STFT v.s. MFCC

STFT (**Short Time Fourier Transform**)



MFCC (**Mel Frequency Cepstral Coefficient**)



We used **STFT** because we obtained better results and our models had a better accuracy. Indeed, as we can see in the images above, the STFT signal is more evident than the other.

Training set & Test set

Both the datasets of STFT and MFCC were divided into training set and test set.
Each dataset was divided in two sections balanced for each class:

- Training set -> 70% of the total images (3276 of 4680)
- Test set -> 30% of the total images (1404 of 4680)

CNNs

To classify the spectrograms, we used convolutional neural networks. We decided to approach the problem gradually increasing the complexity of the networks, using the following layers:

- 2D Convolutional layers;
- Batch Normalization layers;
- Max Pooling 2D layers;
- Dense Fully connected layers;
- Dropout layer.

We applied each model both to the STFT and the MFCC spectrograms.

CNNs

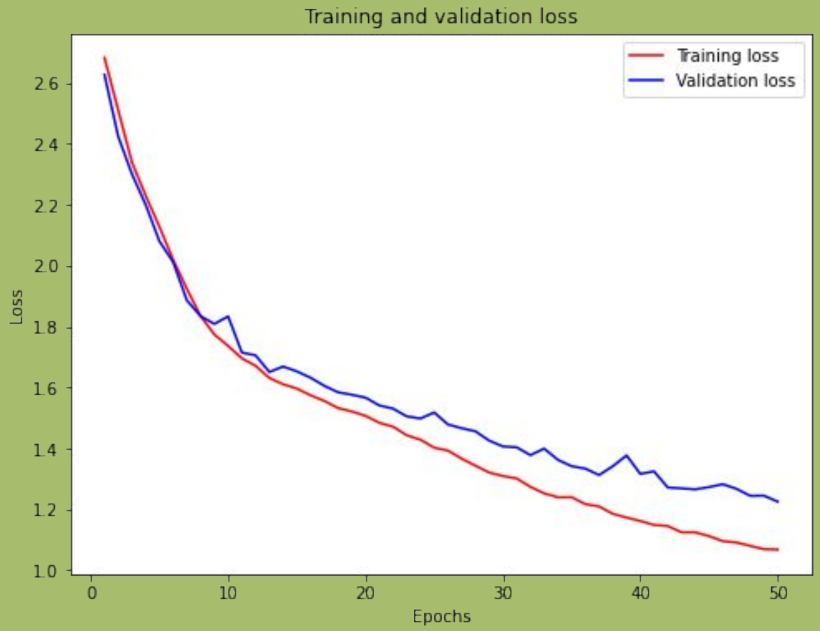
We trained the models for **50 epochs** using early stopping, based on the loss function, after 5 epochs.
We trained all the models using both RMSprop and Adam as optimizers, with **learning rate** equal to **0.001**, finding a general improvement when using Adam optimizer.

We used **sparse categorical crossentropy** as loss function (we have more than 2 classes).

First Model

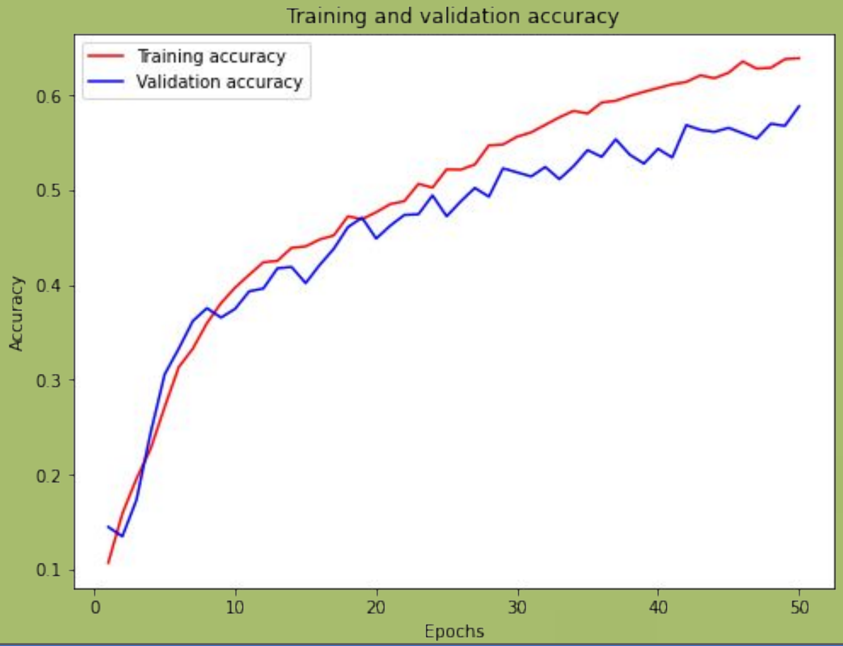
```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Input(shape=(IMAGE_HEIGHT, IMAGE_WIDTH, N_CHANNELS)))

model.add(tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.MaxPooling2D(3, strides=3, padding="same"))

model.add(tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'))

model.add(tf.keras.layers.GlobalMaxPooling2D())

model.add(tf.keras.layers.Dense(N_CLASSES, activation='softmax'))
```

```
=================================================================
Total params: 20,367
Trainable params: 20,367
Non-trainable params: 0
_________________________________________________________________
```



Final Loss: **1.225**

Final accuracy: **58.83%**



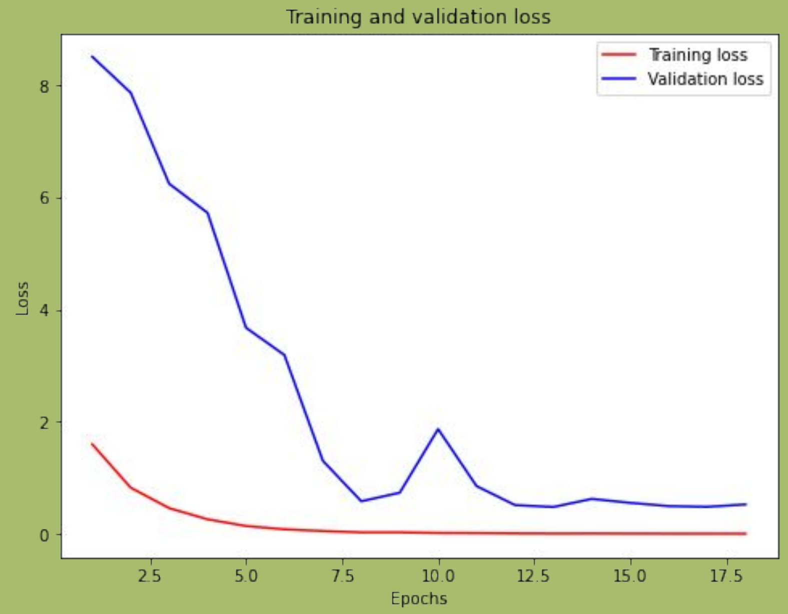
Second Model

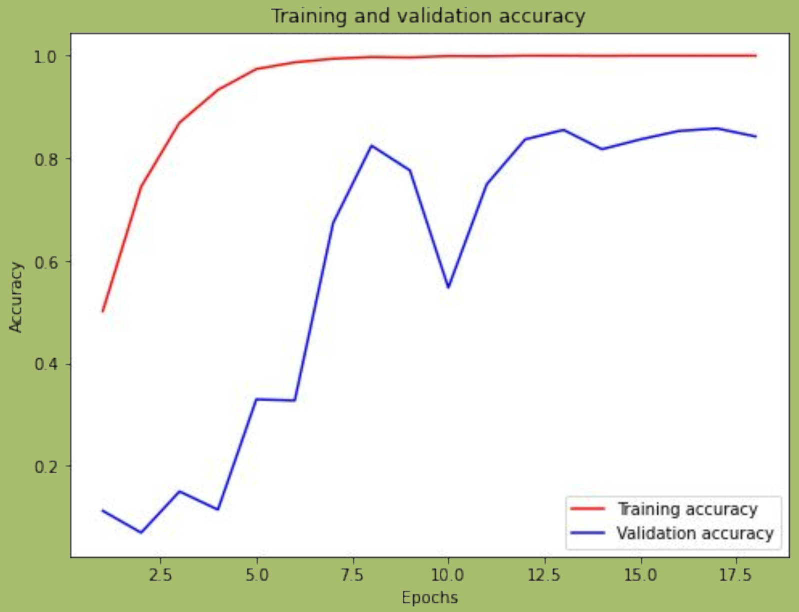
```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Input(shape=(IMAGE_HEIGHT, IMAGE_WIDTH, N_CHANNELS)))

model.add(tf.keras.layers.Conv2D(32, 3, strides=2, padding='same', activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.BatchNormalization())

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(N_CLASSES, activation='softmax'))
```

```
=================================================================
Total params: 8,488,783
Trainable params: 8,487,375
Non-trainable params: 1,408
```

Final Loss: **0.525**

Final accuracy: **84.26%**





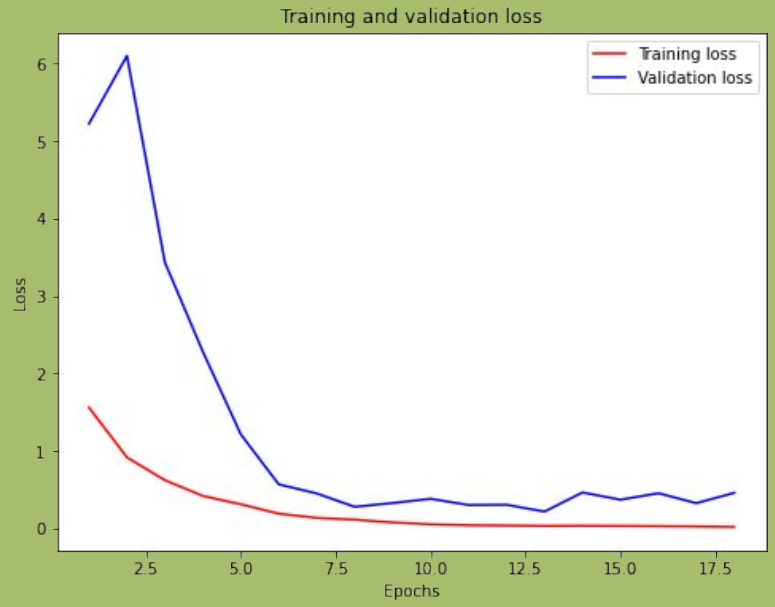
Best Model

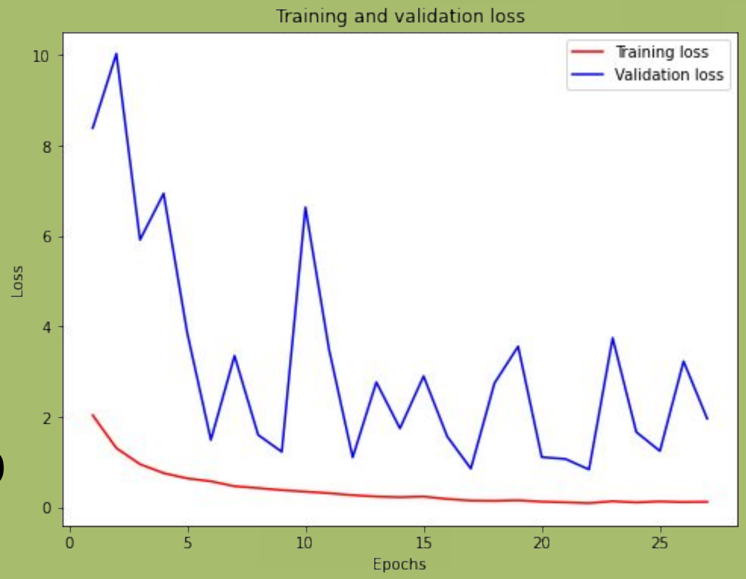
```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Input(shape=(IMAGE_HEIGHT, IMAGE_WIDTH, N_CHANNELS)))
model.add(tf.keras.layers.Conv2D(16, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Conv2D(256, 3, padding='same', activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(N_CLASSES, activation='softmax'))
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
```

🟡	Layers
🔴	STOP!

```
=================================================================
Total params: 4,596,015
Trainable params: 4,593,519
Non-trainable params: 2,496
_________________________________________________________________
```

Best Model



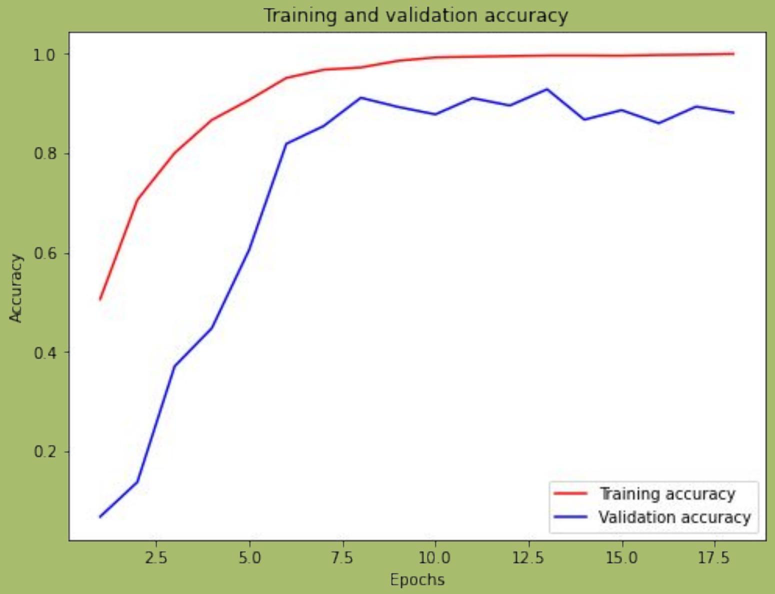


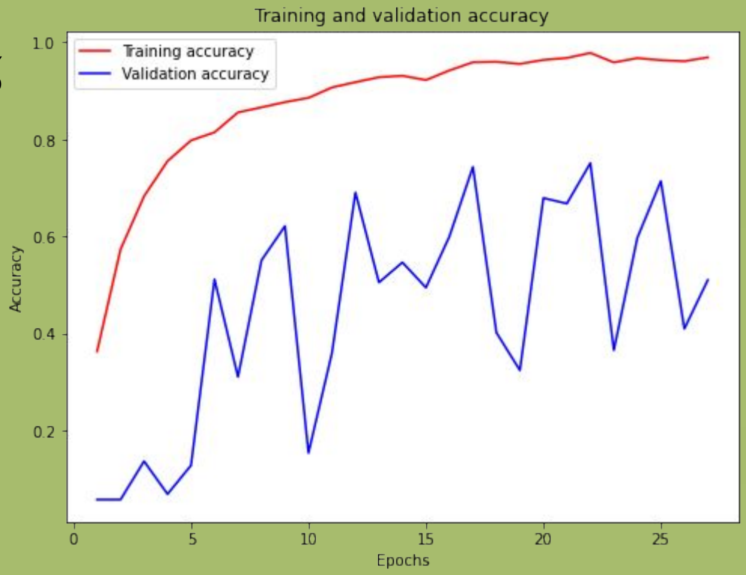
Final Loss: **0.455**

Final Loss: **1.960**

STFT

MFCCs





Final Accuracy: **51.07%**

Final Accuracy: **88.11%**

Future developments

- Expanding the training dataset through data augmentation.
- Improve the value of accuracy through network architecture.

Thank you for your attention!