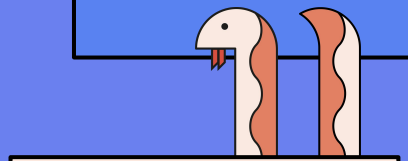
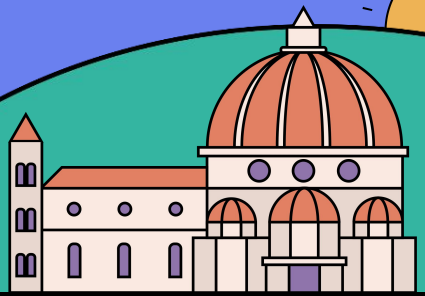
How to use Deep Learning to process audio data and classify sounds



Serena Palazzo
Team Leader AI & Data Science



HOVERTURE
A SYMPHONIE PRIME COMPANY

SYMPHONIE PRIME

Florence 25-28 May 2023

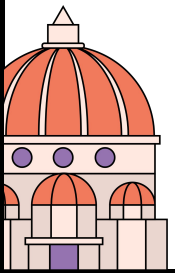
PYCON(IT) 23

SYMPHONIE PRIME

Is a European Group, holding of a series of technological boutiques, united by the mission of developing the **Digital Transformation** of medium and large companies with a view to Open Innovation

Google Cloud Partner

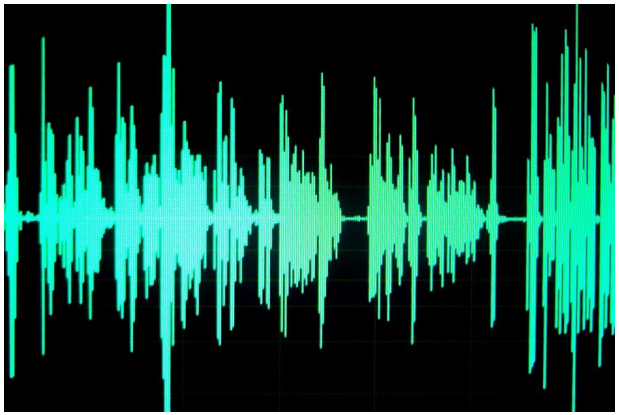
HOVERTURE
A SYMPHONIE PRIME COMPANY

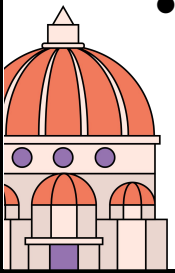


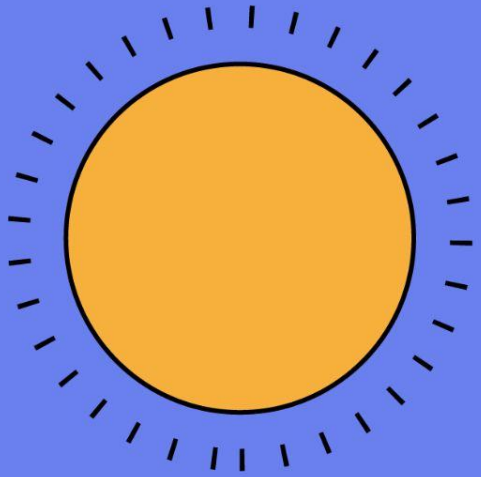
- It is one of the technological boutiques within Symphonie, made of a multicultural team of talents
- We deliver the benefit of the Cloud, through the most innovative and cutting-edge technology
- Google Premier Partner since 2014
- Several years of experience in Cloud Transformation and Big Data

OVERVIEW

- Sound data: **from raw to digital** format
- Audio data **common use cases**
- Audio **data preparation** to feed Deep Learning models
- **Building** a model from scratch for sound classification
- **Transfer Learning** techniques





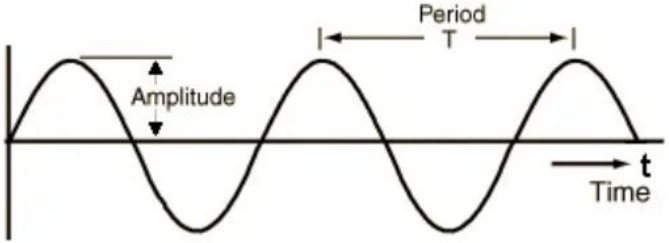


Sound data

What is sound?

Regular sound signal

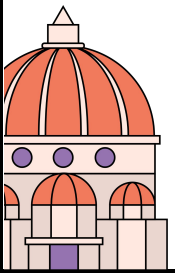
- In its **rawest** (non-digital) form, is produced by variation in air pressure.
- Sound signals often repeat at **regular intervals**



Composite sound signals

- Majority of sounds may not follow regular periodic patterns.
- Signals of different frequencies can be added together to create composite signals with more complex repeating patterns.
  - **A mix of waves, i.e a mix of frequencies at different amplitudes**

The human ear is able to differentiate between different sounds based on the 'quality' of the sound, i.e **timbre**

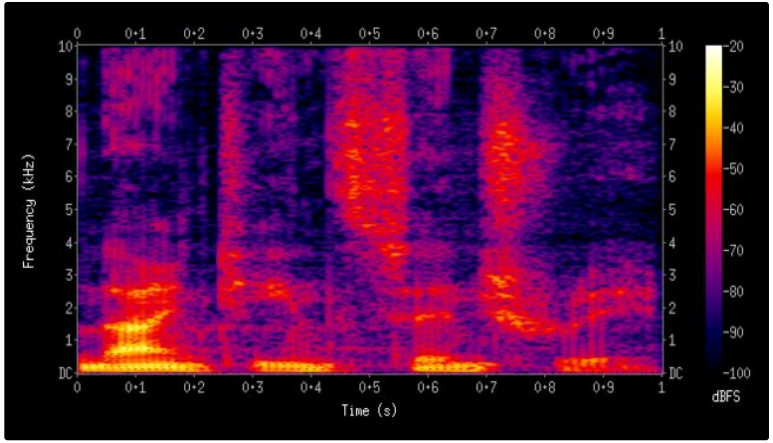


From analogical sound to digital audio

- The **digital form** of sound is Audio, converted in numbers.
- Two important types of numbers: **amplitude** and **frequency**

How to convert sound to digital audio?

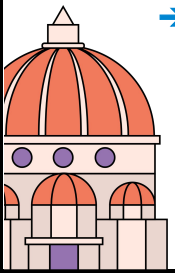
- It happens through **“Fourier Transform”** that decompose complex audio into **spectrograms**



But what is a Spectrogram?

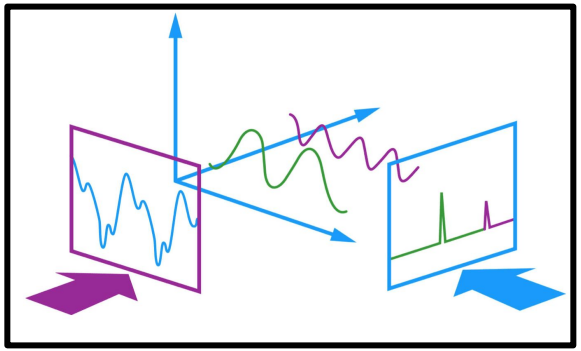
- Spectrograms are images that allows to represent audio over time
- Frequency as a function of time with the color that indicates the amplitude of a specific frequency at a specific point of time

Really? Convert sound into pictures? This sounds like science fiction 😄



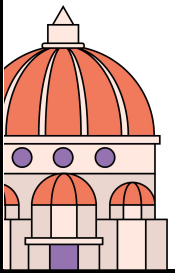
Sound data: which is the challenge?

Sound data and AI: some important points



- ➔ To feed an AI model sound has to be transformed in numbers, this is why is needed a digital transformation
  - Converting **pictures into spectrograms**, i.e. images is one of the common ways to have sound data into a format comprehensive for AI models

- ➔ Sound has a **time factor**.
  - Comparable to video data rather than image
    - because sound fragments is of certain duration rather than just one single point in time.
    - Some of the most suitable architectures to train from scratch: CNN & LSTM

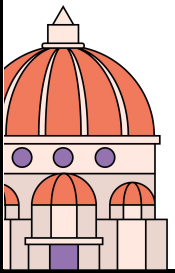


Deep learning landscape for audio applications

Audio data in day-to-day life come in innumerable forms such as human speech, music, animal voices, and other natural sounds etc.

There are a vast number of usage scenarios that require us to process and analyze audio:

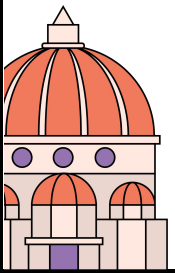
- **Audio classification** (identification of source or type of sound)
- **Audio segmentation** and separation (separate person's voice from background noise)
- **Music genre classification** and tagging
- **Music generation** and **music transcription**
- **Voice recognition** (identify gender or mood from voice)
- **Speech-to-text** and **text-to-speech** (Virtual assistant such as Alexa, Cortana, Siri)



What we will see today with audio data?

- One of the most widely used applications in Audio Deep Learning involves learning to **classify sounds** and to **predict categories** (such as music genre, speakers etc)

- In this context I will show how to classify a dataset of **Urban sound audio** by using different approaches:
  - By Building a Convolutional Neural Network
  - By using the latest state-of-art Transfer Learning techniques



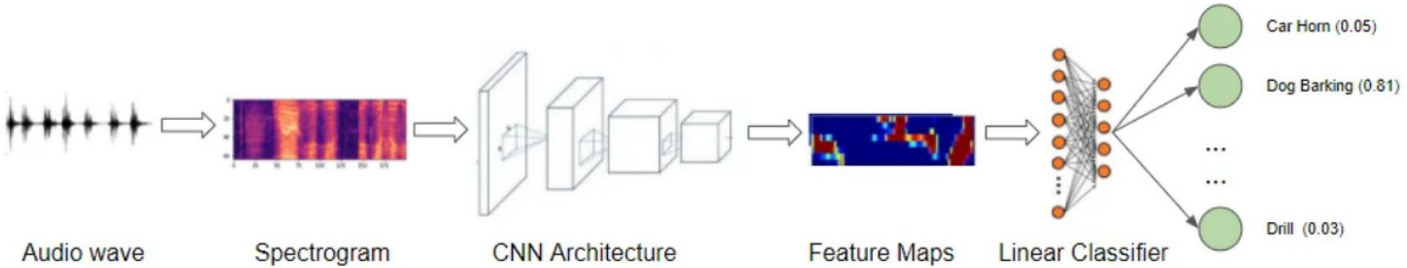
Spectrograms for deep learning

Raw data in the form of wave file

Convert audio data into its corresponding Spectrogram

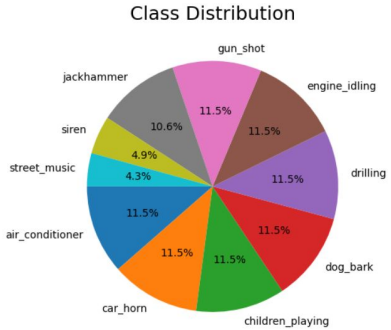
Build a Deep Learning architecture such as CNN

Generate Output predictions



Audio dataset

- The dataset contains 8k sounds recorded from dat-to-day city life
- It has **audio files in .wav format**, the start and the end of the audio, the class label (in numeric and string format) for a total of **10 classes**
- Samples of around 4 seconds in length

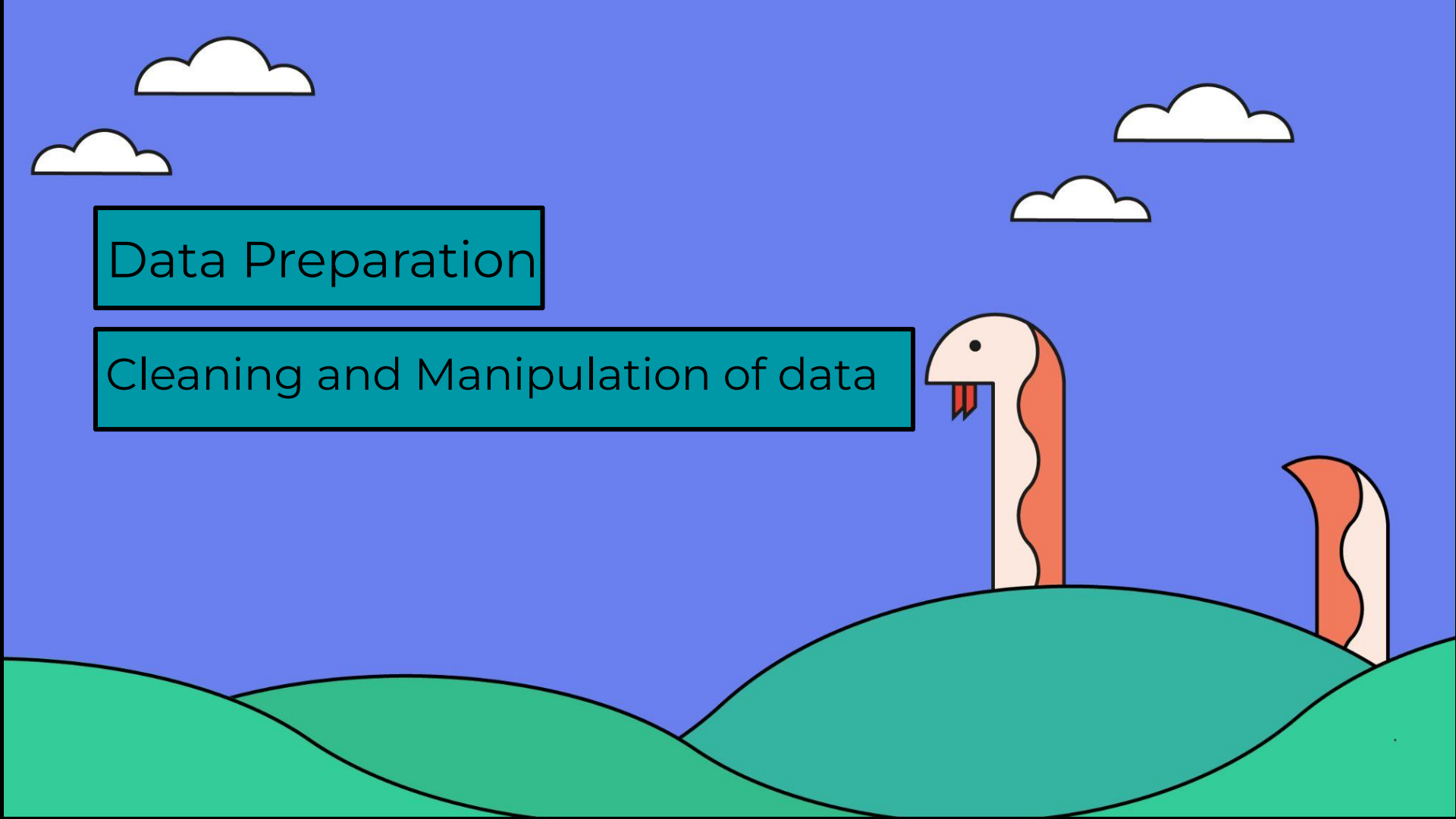



```
import pandas as pd
df = pd.read_csv("UrbanSound8K/metadata/
UrbanSound8K.csv")
df.sample(5)
```

	slice_file_name	fsID	start	end	salience	fold	classID	class
3153	165643-4-0-0.wav	165643	0.000000	4.000000	1	2	4	drilling
5843	24347-8-0-90.wav	24347	49.120371	53.120371	2	4	8	siren
8682	99192-4-0-0.wav	99192	0.000000	4.000000	1	10	4	drilling
6248	36429-2-0-15.wav	36429	7.500000	11.500000	2	8	2	children_playing
3301	168037-4-0-0.wav	168037	1.302542	3.362927	1	8	4	drilling

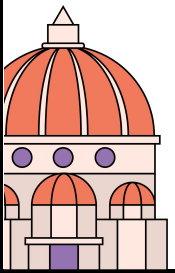
Data Preparation

Cleaning and Manipulation of data



Data preparation

- As well as for tabular, images, text and video data, also audio data need preprocessing
- Python has some great **libraries** for audio processing, to:
  - Read audio files in different formats
  - Visualize sound wave
  - listen to it
  - Transform and augment



Librosa

one of the most popular and with a wide range of features

librosa

Scipy

Commonly used with similar set of features as Librosa

TorchAudio

integrated with Pytorch, specifically for deep learning

TorchAudio

Load audio data: some examples

➔ Load data with **librosa**

```
import librosa

# Load the audio file
audio_file = './track_sound.wav'
samples, sample_rate = librosa.load(audio_file, sr=None)
```

➔ Load data with **scipy**

```
from scipy.io import wavfile

# Load the audio file
audio_file = './track_sound.wav'
sample_rate, samples = wavfile.read(audio_file)
```

Audio data is represented as time series of numbers that represent the **Amplitude at each timestep**

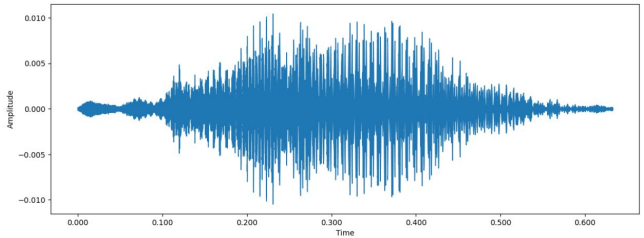
```
print ('Example shape ', samples.shape, 'Sample rate
', sample_rate, 'Data type', type(samples))
print (samples[22400:22420])
```

```
Example shape  (30365,) Sample rate  48000 Data type <class 'numpy.ndarray'>
[-0.00076294 -0.00073242 -0.0007019  -0.00064087 -0.00061035 -0.00057983
 -0.00054932 -0.0005188  -0.0005188  -0.00048828 -0.0005188  -0.00048828
 -0.00045776 -0.0005188  -0.0005188  -0.00061035 -0.0007019  -0.00076294
 -0.00079346 -0.00085449]
```

Visualize and listen to audio

➔ **Visualize** sound wave with librosa

```
import librosa.display
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 5))
librosa.display.waveshow(samples, sr=sample_rate)
```



➔ **Listen** to it, by using Jupyter notebook it is possible to play audio from cell

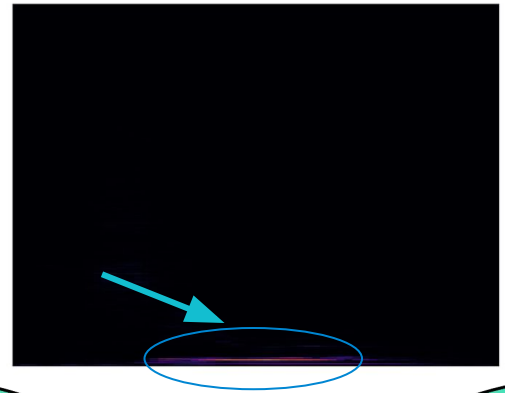
```
from IPython.display import Audio

Audio(audio_file)
```

Convert raw audio in Spectrograms

- Before to feed audio data into Machine Learning or Deep Learning models, a common practice is to convert them into spectrogram
- A spectrogram is a ‘snapshot’ of an audio wave and since it is an image it is well suited to being used as an input for CNN-based architectures

librosa: spectrogram conversion

```
sgram = librosa.stft(samples)
librosa.display.specshow(sgram)
```



- Frequency as a function of Time.
- Different colors indicate the Amplitude of each frequency.
- The brighter is the color the higher the energy of the signal

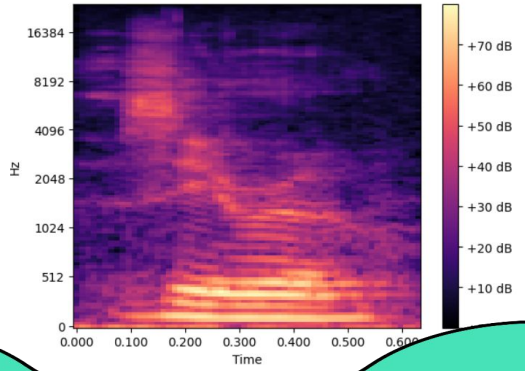
How to deal with sound in a realistic manner?

- ➔ Since human perceive frequencies and amplitude in a **logarithmic scale**, rather than linear, we need to account for this
- ➔ **Mel** and **Decibel** scales are needed to realistically deal with frequencies and amplitudes of sound

How to do with code?

- ➔ Use **Mel scale** instead Frequency on the y-axis
- ➔ Use **Decibel scale** instead of Amplitude to indicate colors

```
# use the mel-scale instead of raw frequency
sgram_mag, _ = librosa.magphase(sgram)
mel_scale_sgram = librosa.feature.melspectrogram(S=sgram_mag, sr=sample_rate)

# use the decibel scale to get the final Mel Spectrogram
mel_sgram = librosa.amplitude_to_db(mel_scale_sgram, ref=np.min)
librosa.display.specshow(mel_sgram, sr=sample_rate, x_axis='time', y_axis='mel')
plt.colorbar(format='%+2.0f dB')
```

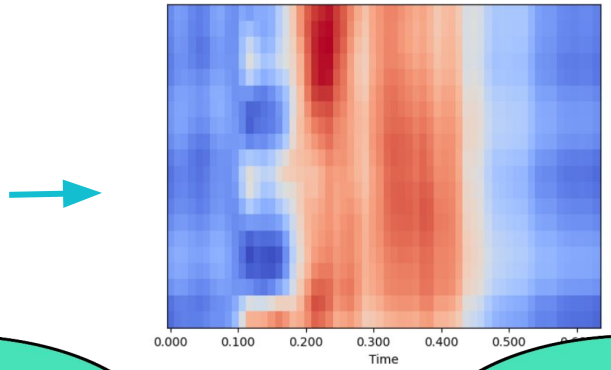


Spectrogram optimization

- To get the **best performances** for our models, Mel Spectrograms need to be optimized
- To deal with human speech, **MFCC** (Mel Frequency Cepstral Coefficients) works better
  - The MFCC extracts a much smaller set of features from the audio that are the most relevant in capturing the essential quality of the sound.

librosa: MFCC

```
import librosa.display

mfcc = sklearn.preprocessing.scale(mfcc, axis=1)
librosa.display.specshow(mfcc, sr=sample_rate, x_axis='time')

print (f'MFCC is of type {type(mfcc)} with shape {mfcc.shape}')
MFCC is of type <class 'numpy.ndarray'> with shape (20, 60)
```

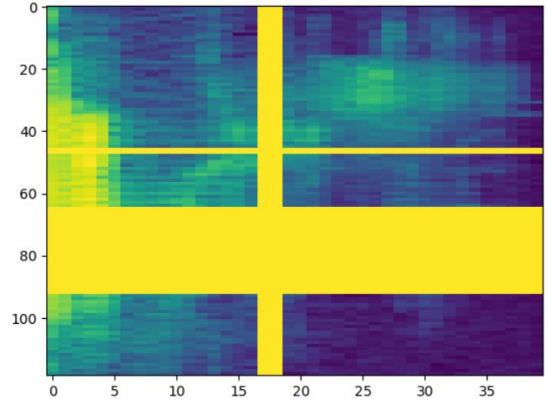


Spectrogram augmentation

- A common technique used to increase the diversity of your dataset, when data are not enough, is to **augment data artificially**.
- Augment data means **make modifications** to the available data
  - For the model these modifications looks like new data and this allow to generalize better

SpecAugment for audio data

- SpecAugment, method used for audio data, act on:
  - **Frequency mask**: randomly mask out a range of consecutive frequencies by adding horizontal bars on the spectrogram
  - **Time mask**: randomly block out ranges of time from the spectrogram by using vertical bars



- For **raw data** the augmentation can act by adding silence, add noise, speed up or slow down sound, modify frequency.

Training data

Data preparation: Encoding, split, reshape

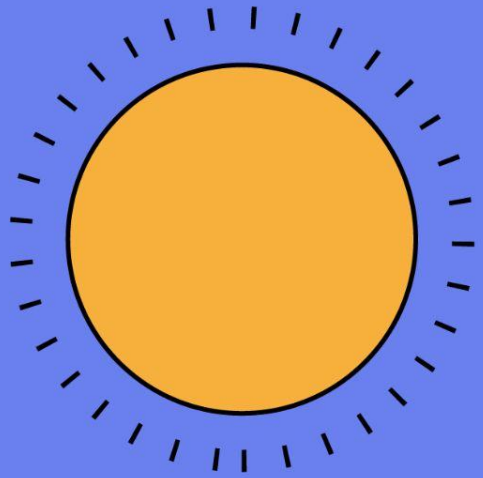
1) **Label encoding** of target values to convert classes into categories

```
from sklearn.preprocessing import LabelEncoder

# Encode the classification labels
le = LabelEncoder()
y = to_categorical(le.fit_transform(y))
```

2) **Split** into train, test and validation dataset (80% train, 10% test, 10% val)

```
from sklearn.model_selection import train_test_split
# Split the dataset:
X_train, X_temp, y_train, y_temp = train_test_split(X, y, train_size=0.8,
random_state = 3)
X_val, X_test, y_val, y_test =  train_test_split(X_temp, y_temp, train_size=0.5,
random_state = 3)
```

3) **Reshape** to ensure the correct shape before to feed data into the model

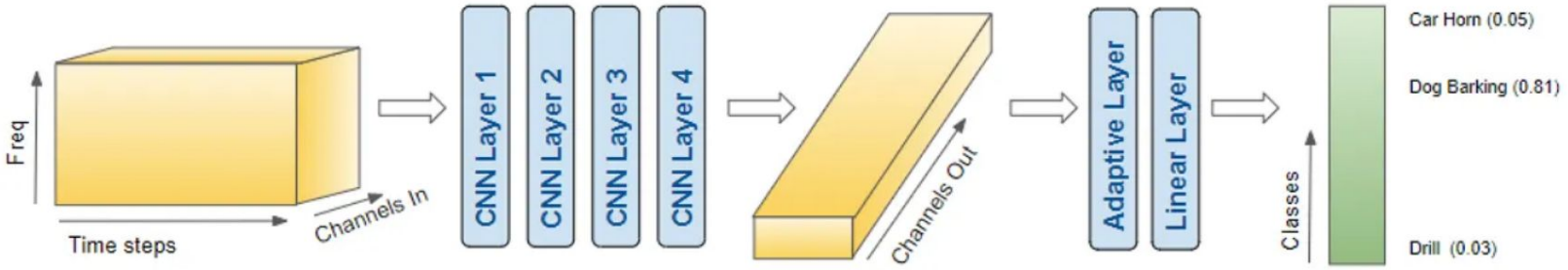


AI MODELS

Deep Learning model

AI MODEL

- Since data consists now of Spectrogram images, **Convolutional Neural Networks (CNN)** classification architectures are used to process these images.
  - CNN are particularly suitable to process pixel data
  - To build our CNN, the Python Keras library has been used



CNN: Model building

- Keras has a **model class** that includes three ways to create a model;
- In our case we choose the **Sequential model**;
  - This method allows us to add layers to the architecture;
- Keras **compile method** is used to configure the model training and to set the **hyperparameters** of the model;

```
# Constructing model with RELu and SoftMax activation functions:

model_relu = Sequential()
model_relu.add(Conv2D(filters=16, kernel_size=2, input_shape=(num_rows,
num_columns, num_channels), activation='relu'))
model_relu.add(MaxPooling2D(pool_size=(2,2)))
model_relu.add(Dropout(0.2))

model_relu.add(Conv2D(filters=32, kernel_size=2, activation='relu'))
model_relu.add(MaxPooling2D(pool_size=(2,2)))
model_relu.add(Dropout(0.2))

model_relu.add(Conv2D(filters=64, kernel_size=2, activation='relu'))
model_relu.add(MaxPooling2D(pool_size=(2,2)))
model_relu.add(Dropout(0.2))

model_relu.add(Conv2D(filters=128, kernel_size=2, activation='relu'))
model_relu.add(MaxPooling2D(pool_size=(2,2)))
model_relu.add(Dropout(0.2))

model_relu.add(GlobalAveragePooling2D())
model_relu.add(Flatten())
model_relu.add(Dense(num_labels, activation='softmax'))

model_relu.compile(optimizer='adam', loss='categorical_crossentropy',
                   metrics=['accuracy'])
```

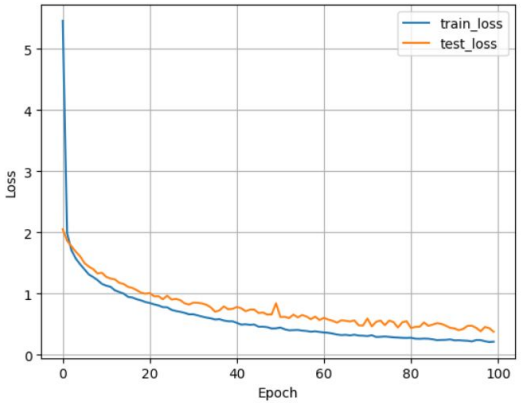
CNN: Model training

➔ Keras **fit method** is used to train the model for a fixed number of **epochs** (iterations on a dataset).

```
history_relu = model_relu.fit(X_train, y_train, batch_size=256, epochs=100,
validation_data = (X_test, y_test), callbacks=[checkpointer], verbose=1)
```

```
Epoch 1/100
28/28 [==============================] - ETA: 0s - loss: 5.4574 - accuracy: 0.1754
Epoch 1: val_loss improved from inf to 2.05036, saving model to saved_models/weights.best.basic_cnn.hdf5
28/28 [==============================] - 15s 496ms/step - loss: 5.4574 - accuracy: 0.1754 - val_loss: 2.0504 - val_accuracy: 0.2357
Epoch 2/100
28/28 [==============================] - ETA: 0s - loss: 1.9862 - accuracy: 0.3008
Epoch 2: val_loss improved from 2.05036 to 1.86370, saving model to saved_models/weights.best.basic_cnn.hdf5
28/28 [==============================] - 14s 493ms/step - loss: 1.9862 - accuracy: 0.3008 - val_loss: 1.8637 - val_accuracy: 0.3513
Epoch 3/100
28/28 [==============================] - ETA: 0s - loss: 1.7083 - accuracy: 0.3894
Epoch 3: val_loss improved from 1.86370 to 1.77323, saving model to saved_models/weights.best.basic_cnn.hdf5
28/28 [==============================] - 15s 535ms/step - loss: 1.7083 - accuracy: 0.3894 - val_loss: 1.7732 - val_accuracy: 0.4085
Epoch 4/100
28/28 [==============================] - ETA: 0s - loss: 1.5717 - accuracy: 0.4451
Epoch 4: val_loss improved from 1.77323 to 1.68390, saving model to saved_models/weights.best.basic_cnn.hdf5
28/28 [==============================] - 14s 491ms/step - loss: 1.5717 - accuracy: 0.4451 - val_loss: 1.6839 - val_accuracy: 0.4497
Epoch 5/100
28/28 [==============================] - ETA: 0s - loss: 1.4785 - accuracy: 0.4936
Epoch 5: val_loss improved from 1.68390 to 1.60874, saving model to saved_models/weights.best.basic_cnn.hdf5
28/28 [==============================] - 14s 495ms/step - loss: 1.4785 - accuracy: 0.4936 - val_loss: 1.6087 - val_accuracy: 0.4634
Epoch 6/100
28/28 [==============================] - ETA: 0s - loss: 1.3951 - accuracy: 0.5132
Epoch 6: val_loss improved from 1.60874 to 1.49796, saving model to saved_models/weights.best.basic_cnn.hdf5
28/28 [==============================] - 14s 487ms/step - loss: 1.3951 - accuracy: 0.5132 - val_loss: 1.4980 - val_accuracy: 0.5343
Epoch 7/100
28/28 [==============================] - ETA: 0s - loss: 1.3120 - accuracy: 0.5389
Epoch 7: val_loss improved from 1.49796 to 1.44021, saving model to saved_models/weights.best.basic_cnn.hdf5
28/28 [==============================] - 14s 517ms/step - loss: 1.3120 - accuracy: 0.5389 - val_loss: 1.4402 - val_accura
Epoch 8/100
```

Loss function does not show any overfitting



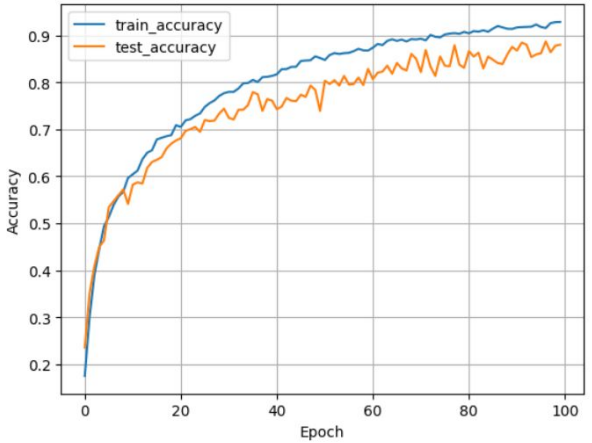
Model performances assessment

Evaluation of model metrics

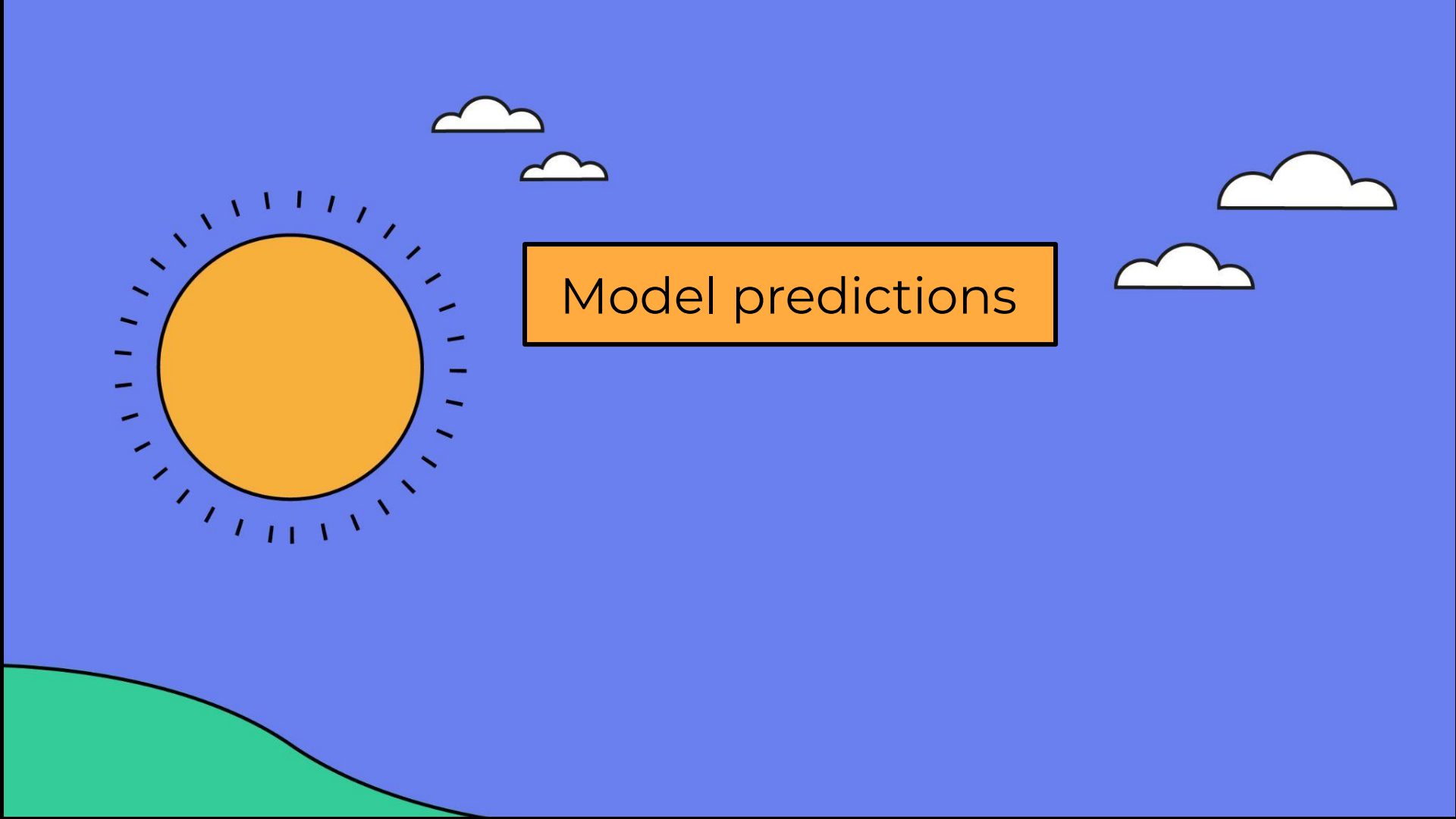
Evaluation Metrics

- Measuring the performance of any AI model is very important to assess the accuracy of the model;
- There are different types of evaluation metrics used for the AI and the choice depends on the model used to generate the result;

```
# Evaluating the model on the training and testing set:
score = model_relu.evaluate(X_train, y_train, verbose=0)
print("Training Accuracy: ", score[1])

score = model_relu.evaluate(X_test, y_test, verbose=0)
print("Testing Accuracy: ", score[1])
```



Training Accuracy: 0.9609162211418152
Testing Accuracy: 0.8798627257347107


Model predictions

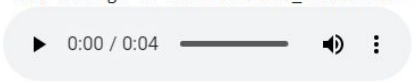
Predictions

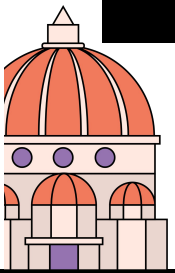
➔ Once a model is trained and evaluated, it is ready to make predictions on a new set of unseen data;

```
def prediction_(path_file):
    data_sound = extract_features(path_file)
    X = np.array(data_sound)
    pred_result =
model_relu.predict(X.reshape(1,40,174,1))
    pred_class = pred_result.argmax()
    pred_prob = pred_result.max()
    print(f"This belongs to class {pred_class} :
{classes[int(pred_class)]}  with {pred_prob} probility %")
```

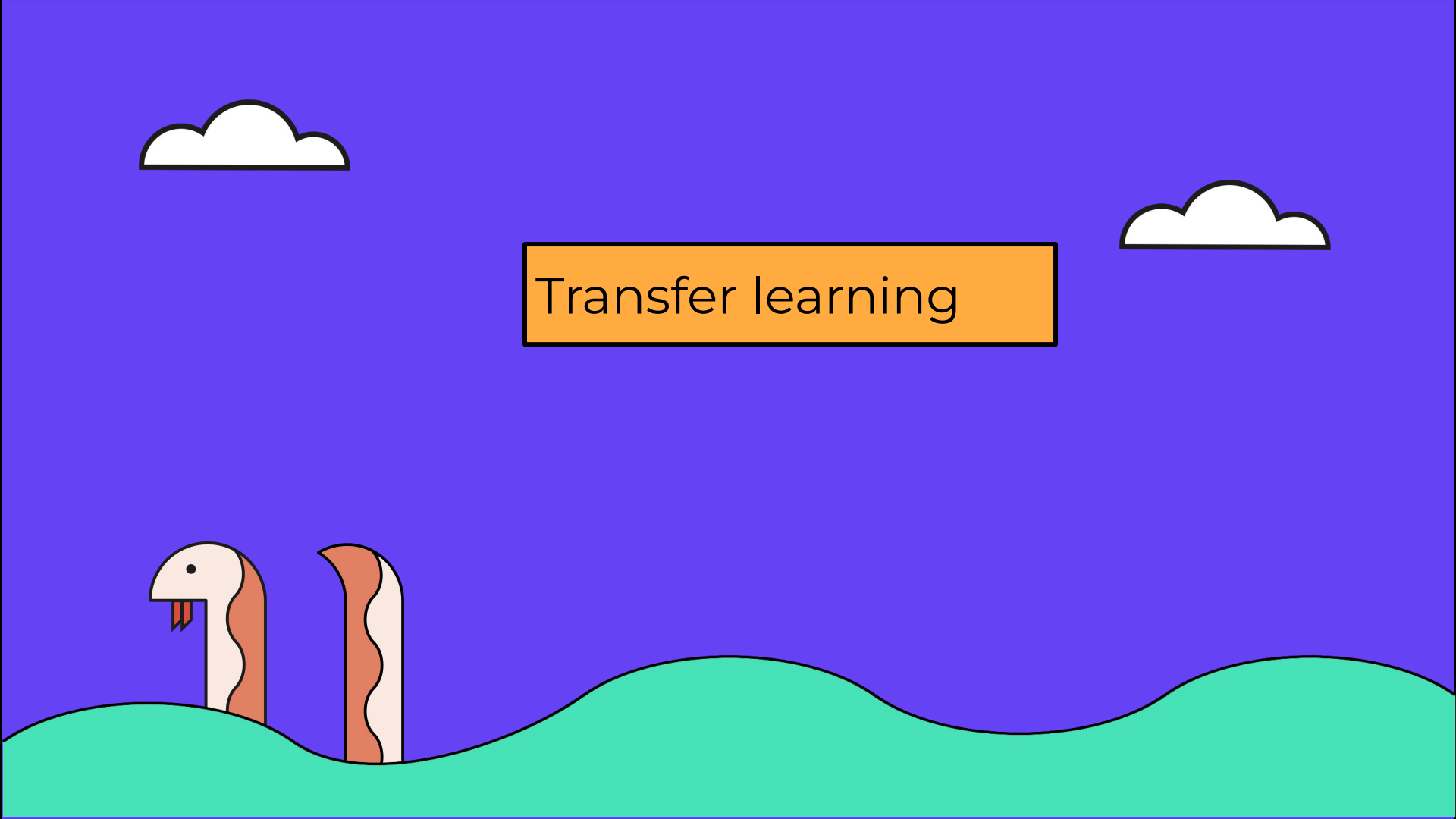
```
prediction_(metadata.path_file[500])
ipd.Audio(metadata.path_file[500])
```

```
This belongs to class 0 : air_conditioner  with 0.9854398369789124 probility %
```





Transfer learning

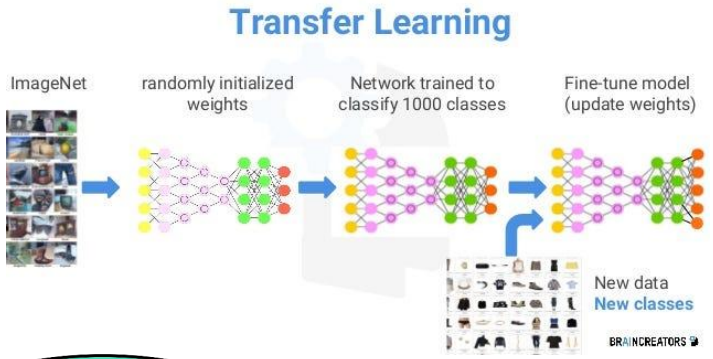


Transfer learning overview

- ➔ Consists of retrain a Network that has been already trained (pre-trained) with a huge dataset to perform a fine-tuning of the model by using our own dataset
  - The type input is the same (if a Network has been trained on images the new input has to be image dataset)
  - The idea is to leverage already learned features without train from scratch with a huge amount of data

Why use transfer learning?

- ➔ Save time and resource
- ➔ Have more accurate results by leveraging on the pretrained features learned



How to use Transfer learning for audio

- **Hugging Face** is an AI community, a large open-source community
- It provides state-of-the-art models for different tasks, mostly based on **Transformers**
  - It has a vast number of pre-trained models, in particular for Image, Text and audio tasks
  - Some of these pretrained models are **Transformers acoustic models**

Common models for audio

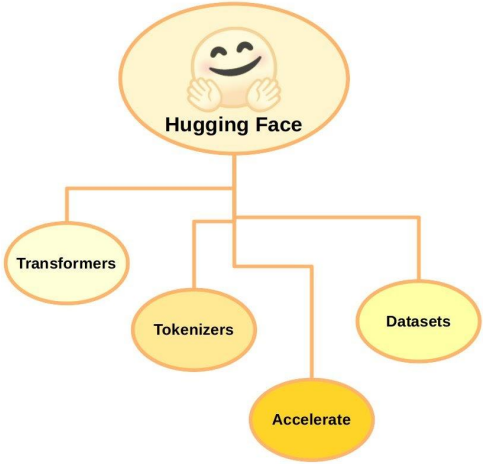
- Wav2Vec (from facebook, available in Hugging Face)
- YamNet (from tensorflow Hub)
- SpeechBrain (available in Hugging Face)

HUGGING FACE

Fine tuning with 🤗 Transformers: overview

➔ Choose the pretrained model with the **AutoModelForAudioClassification** method

```
from transformers import AutoModelForAudioClassification, TrainingArguments, Trainer
num_labels = len(id2label)
model = AutoModelForAudioClassification.from_pretrained(
   "facebook/wav2vec2-base",
    num_labels=num_labels,
    label2id=label2id,
    id2label=id2label,)
```



➔ Define the **Training arguments** and **evaluation strategy**

```
args = TrainingArguments(f"{model_name}-finetuned-ks", evaluation_strategy =
"epoch", save_strategy = "epoch",learning_rate=3e-5,
per_device_train_batch_size=batch_size, gradient_accumulation_steps=4,
per_device_eval_batch_size=batch_size, num_train_epochs=5, warmup_ratio=0.1,
logging_steps=10, load_best_model_at_end=True, metric_for_best_model="accuracy")
```

Fine tuning with 🤗 Transformers

➔ **Train** and **evaluate** the model to make then predictions

```
trainer = Trainer(
    model,
    args,
    train_dataset=encoded_dataset["train"],
    eval_dataset=encoded_dataset["validation"],
    tokenizer=feature_extractor,
    compute_metrics=compute_metrics)
trainer.train()
```

```
***** Running training *****
  Num examples = 51094
  Num Epochs = 5
  Instantaneous batch size per device = 32
  Total train batch size (w. parallel, distributed & accumulation) = 128
  Gradient Accumulation steps = 4
  Total optimization steps = 1995
```

[1995/1995 1:11:18, Epoch 4/5]

Epoch	Training Loss	Validation Loss	Accuracy
0	0.790800	0.677599	0.900853
1	0.320200	0.206119	0.976317
2	0.221000	0.125691	0.978523
3	0.177300	0.099014	0.981318
4	0.172900	0.095161	0.982348

```
def compute_metrics(eval_pred):
    """Computes accuracy on a batch of predictions"""
    predictions = np.argmax(eval_pred.predictions, axis=1)
    return metric.compute(predictions=predictions, references=eval_pred.label_ids)
trainer.evaluate()
```

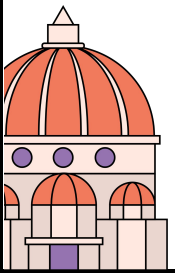
```
***** Running Evaluation *****
  Num examples = 6798
  Batch size = 32
```

[213/213 01:07]

```
{'epoch': 5.0,
 'eval_accuracy': 0.9823477493380406,
 'eval_loss': 0.09516120702028275,
 'eval_runtime': 68.0133,
 'eval_samples_per_second': 99.951,
 'eval_steps_per_second': 3.132}
```

CONCLUSIONS

- We have seen how to use the AI models (both to train from scratch and pre-trained) to be applied for Audio data use cases
- Python brings together all the libraries necessary to carry on an AI project focused on Audio data
  - From data exploration to visualization and then from model building to performance assessment and predictions.



THANKS!