# Unleashing the Power of Transformers in Audio Data Analysis

The rapid advancement of deep learning models has revolutionized various fields, including natural language processing, speech recognition and computer vision. One such groundbreaking model is the Transformer, initially introduced for machine translation tasks. However, its capabilities extend far beyond text-based applications. In recent years, researchers and practitioners have successfully applied transformers to the analysis of audio data, opening up new possibilities such as speech recognition, music generation, and speech classification. This blog explores the possibilities of Transformers in audio and their amazing achievements.

**Transformers:** Transformers, introduced in the seminal paper " Attention Is All You Need" by Vaswani et al., represents a neural network architecture based on self-awareness mechanisms. They have been remarkably successful in capturing long-range dependencies and contextual relationships in sequential data. Instead of using recurrent or convolutional layers, Transformers leverage self-attention mechanisms to weigh the relevance of different parts of the input sequence when generating representations.

## Architecture of Transformers:

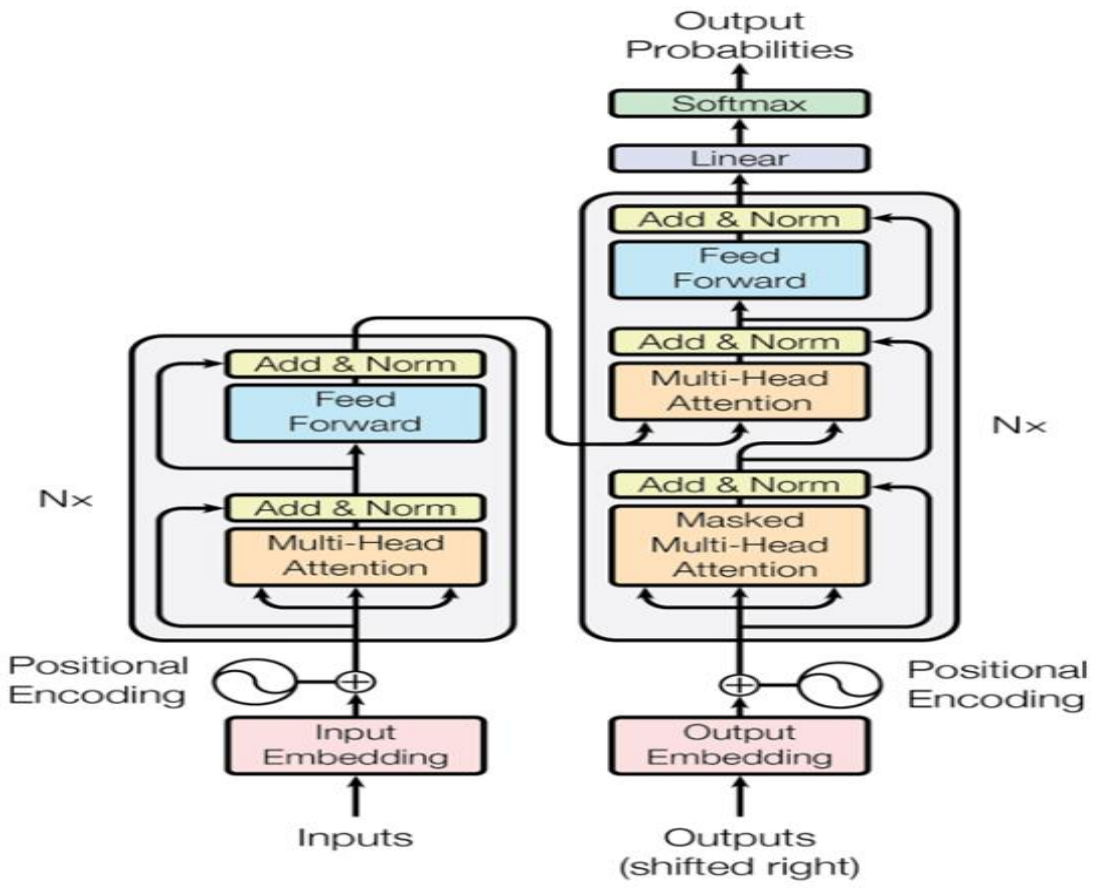
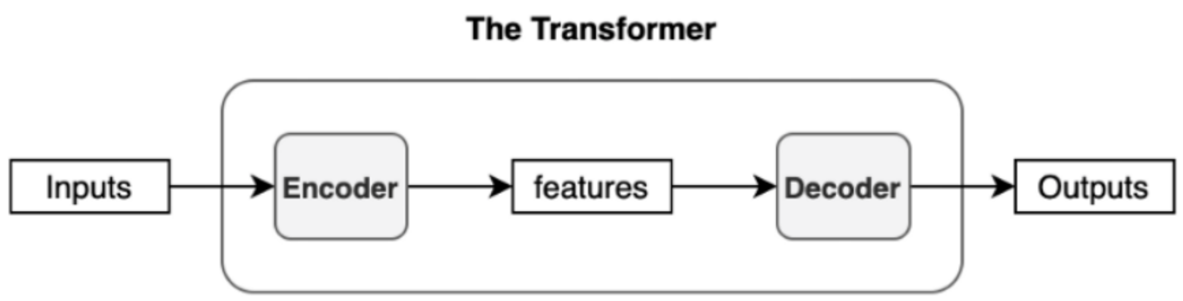
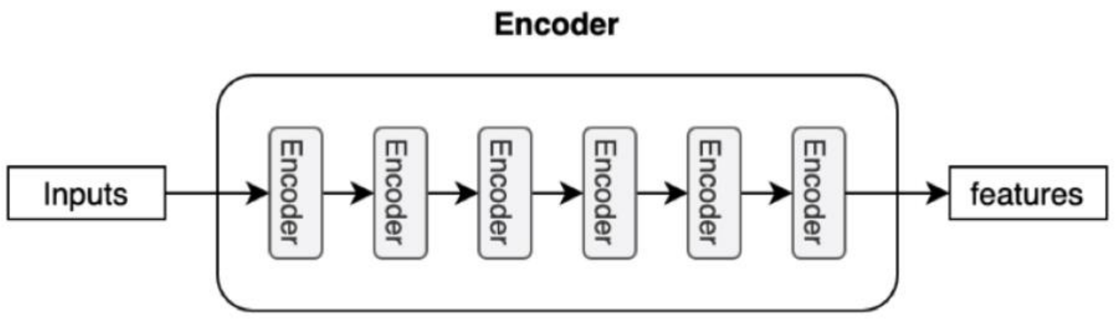


figure1

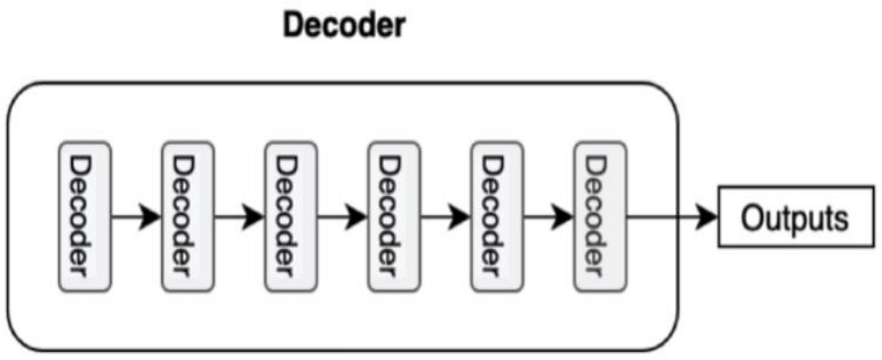


reference for this figure

The function of each encoding layer is to generate encodings that contain information about which parts of the input are related to each other. It passes its encoding to the next encryption layer as input. Each decoder layer does the opposite, taking all the encodings and using their embedded context information to generate the output string. For this, each encryption and decryption layer uses an attention mechanism.



reference for this figure



reference for this figure

For each part of the input, attention considers the relevance of all the other parts and relies on them to produce the output. Each decoder layer has an additional attention mechanism to get information from the output of previous decoders, before the decoder layer gets information from the encoders.

Both the encryption and decryption layers have a forward neural network for additional output processing and contain the remaining connections and layer normalization steps.

## What is Audio Data?

Digital information that represents sound is referred to as audio data. It can be captured using a variety of devices, including microphones, instruments, or sound effects. It can then be processed and stored in a variety of formats, including WAV, MP3, or AAC. Speech recognition, audio processing, and the creation of music and videos are just a few of the many uses for audio data.

## How Audio data feature extraction is carried out for input to Neural networks?

Extracting features from audio data for input to a neural network transforms the raw audio signal into a compact and representative feature representation. This process typically involves applying techniques such as the short-time Fourier transform (STFT), which transforms the audio signal from the time domain to the frequency domain, and the mel-frequency cepstrum coefficients (MFCC), which captures frequency information on the mel-scale. Spectrograms visualize the frequency content of an audio signal over time, mel-spectrograms represent frequencies in the melscale, and chroma functions represent pitches and notes. These feature representations condense relevant information from the audio signal and provide a more meaningful representation for processing by neural networks. Once the features are extracted, they can be fed into various types of neural network architectures such as convolutional neural networks (CNN), transformer neural network (TNN) and recurrent neural networks (RNN) to perform tasks such as speech recognition, music classification, and sound event detection.

## Why do we use transformers for audio data analysis?

Transformers are utilized in audio data analysis due to their exceptional ability to capture long-range dependencies, model complex patterns, and process sequential data efficiently. With self-attention mechanisms, transformers can simultaneously consider all elements in the audio sequence, enabling them to grasp global interactions effectively. Their contextual understanding and parallelization capabilities make them suitable for tasks such as speech recognition, music generation, sound classification, and audio captioning. Moreover, transfer learning with pre-trained transformer models and their success in multimodal processing further enhance their utility in audio analysis, offering state-of-the-art performance and the potential for groundbreaking advancements.

## Applying Transformer to Audio Data:

Audio data, such as speech, music, or environmental sounds, is inherently sequential and time dependent. By leveraging the power of transformers, we can extract meaningful features, analyze audio content, and perform various tasks on audio data. Here are a few areas where transformers have made significant contributions:

1. **Speech Recognition:** Speech recognition systems, which convert spoken language into written text, have greatly benefited from transformer models. Transformers can capture long-range dependencies in audio sequences, making them particularly effective in transcribing speech with improved accuracy. Advanced architectures like the "Listen, Attend and Spell" (LAS) model have achieved state-of-the-art results by incorporating transformers.
2. **Music Generation:** Transformers have also been employed in generating music. By training on large datasets of musical compositions, transformers can learn the underlying patterns and structures of different musical genres. They can generate novel melodies, harmonies, and even entire musical pieces. The Transformer-based music generation models, such as "MuseNet," have impressed listeners with their ability to compose original music.

3. **Sound Classification and tagging:** Transformers have demonstrated exceptional performance in sound classification tasks. By treating audio samples as sequences, transformers can analyze the spectrograms or raw waveform data and classify sounds into various categories, such as human speech, animal sounds, or musical instruments. The models can be trained on large, labeled datasets and effectively learn to recognize audio patterns.

4. **Audio Captioning:** Transformers are also instrumental in generating textual descriptions or captions for audio clips. By incorporating both audio and textual data, transformers can learn to associate specific sounds with their corresponding meanings. This capability finds applications in creating automated systems that provide audio descriptions for visually impaired individuals, enhancing accessibility for multimedia content.

5. **Audio Source Separation:** Transformers have also shown promise in audio source separation, which involves isolating individual sound sources from a mixture of sounds. This task is particularly challenging, but Transformers excel at learning complex patterns and relationships in audio data. By training on mixed audio samples along with their separated sources, Transformers can learn to disentangle overlapping sounds and reconstruct individual sources, contributing to enhanced audio editing and noise reduction techniques.

Let’s take an example of audio classification and see how to use a transformer for audio classification.

In this example, we demonstrate how to fully train the Wav2Vec 2.0 (base) model on the keyword identification task using the Hugging Face Transformers library to obtain cutting-edge results on the Google Speech Commands Dataset.

**Installing all the requirement**

```
!pip install transformers
```

```
!pip install git+https://github.com/huggingface/transformers.git
!pip install datasets
!pip install huggingface-hub
!pip install joblib
!pip install librosa
```

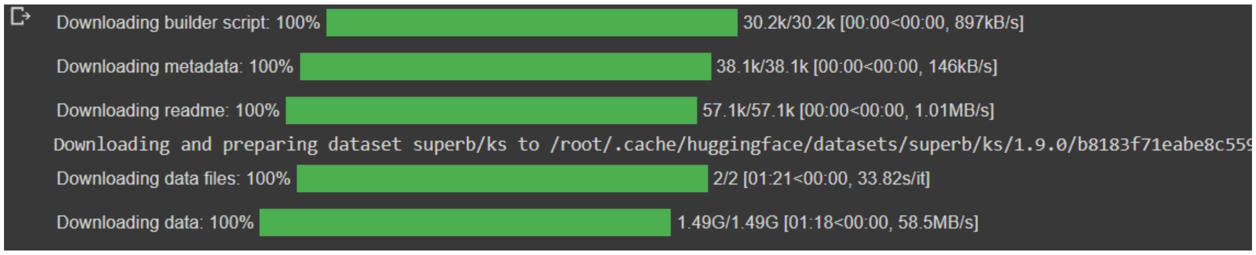
## Importing all required library

```
[ ]
import random
import logging

import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Only log error messages
tf.get_logger().setLevel(logging.ERROR)
# Set random seed
tf.keras.utils.set_random_seed(42)
```

## Loading Google Speech Command Dataset from hugging face dataset

Now we will download the Google Speech Commands V1 Dataset**.** The dataset consists of a total of 60,973 audio files, each of 1 second duration, divided into ten classes of keywords ("Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", and "Go"), a class for silence, and an unknown class to include the false positive. We load the dataset from Hugging Face Datasets. This can be easily done with the load_dataset function.

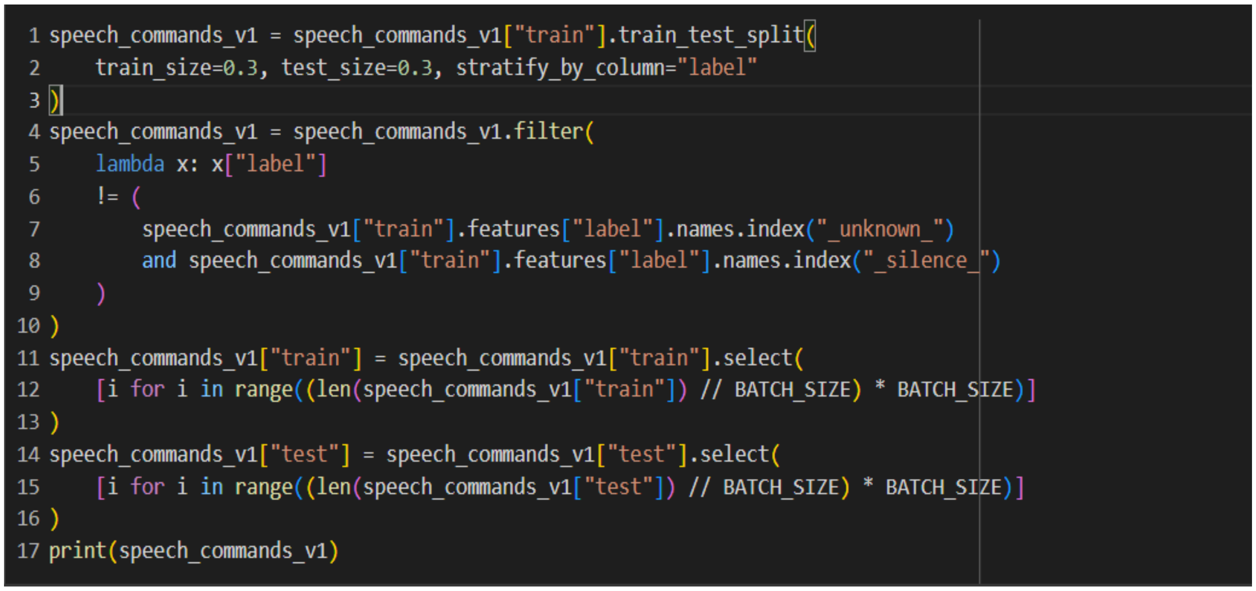
```
from datasets import load_dataset
speech_commands_v1 = load_dataset("superb", "ks")
```



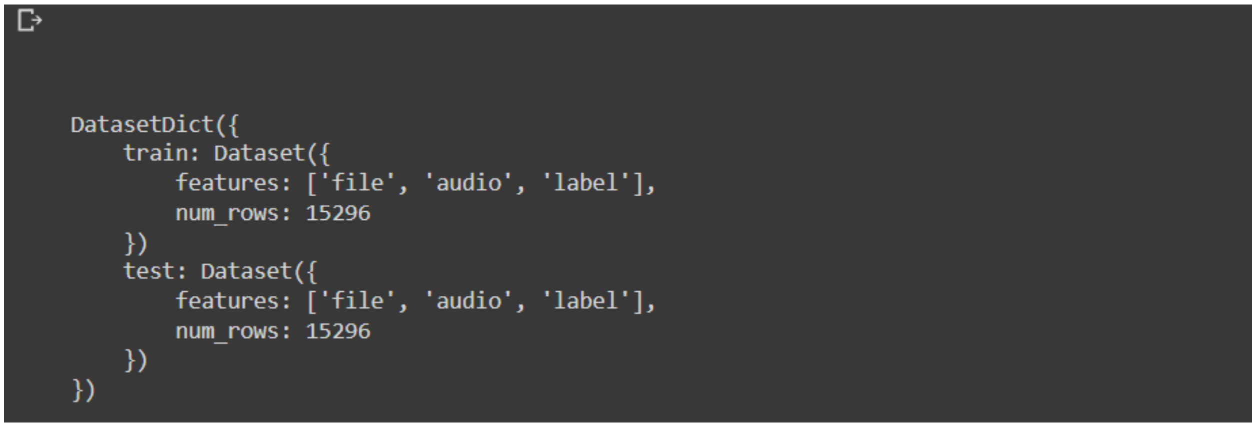
```
Downloading builder script: 100%   30.2k/30.2k [00:00<00:00, 897kB/s]
Downloading metadata: 100%   38.1k/38.1k [00:00<00:00, 146kB/s]
Downloading readme: 100%   57.1k/57.1k [00:00<00:00, 1.01MB/s]
Downloading and preparing dataset superb/ks to /root/.cache/huggingface/datasets/superb/ks/1.9.0/b8183f71eabe8c55
Downloading data files: 100%   2/2 [01:21<00:00, 33.82s/it]
Downloading data: 100%   1.49G/1.49G [01:18<00:00, 58.5MB/s]
```

## Data Pre-processing

Here we will use only 30% of the train and test data sets in order to illustrate the approach. We can easily separate the dataset using the train_test_split method, which expects the split size and the name of the column relative to which you want to stratify.



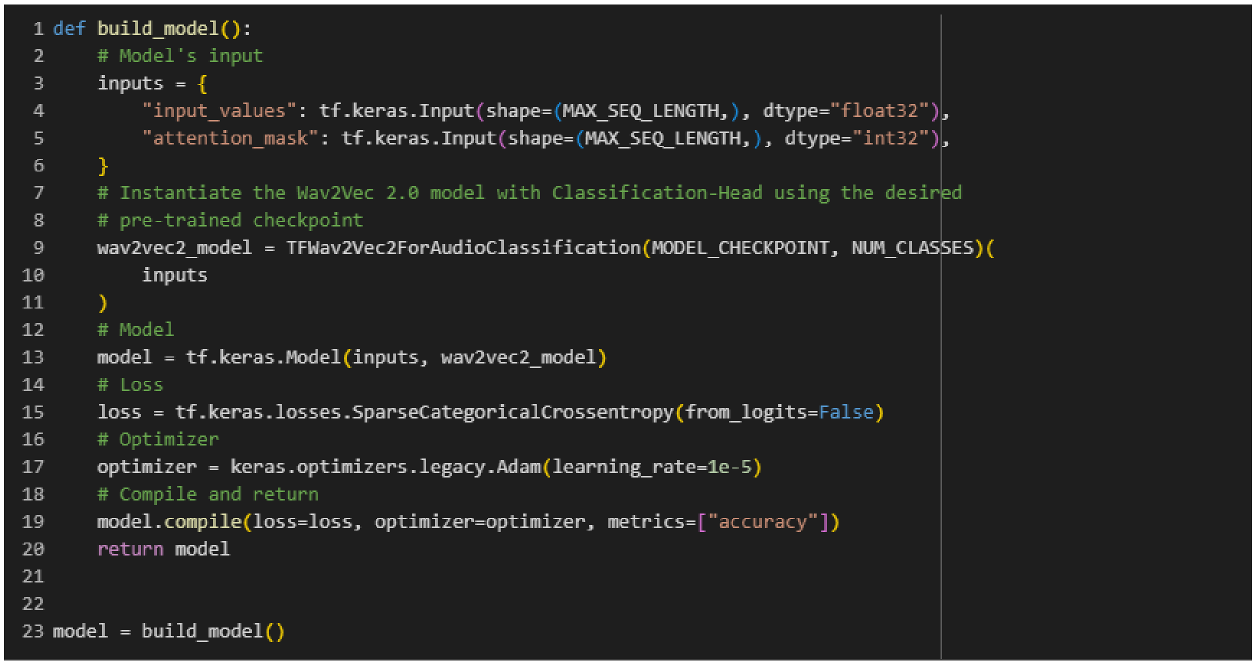
```
speech_commands_v1 = speech_commands_v1["train"].train_test_split(
    train_size=0.3, test_size=0.3, stratify_by_column="label"
)
speech_commands_v1 = speech_commands_v1.filter(
    lambda x: x["label"]
    != (
        speech_commands_v1["train"].features["label"].names.index("_unknown_")
        and speech_commands_v1["train"].features["label"].names.index("_silence_")
    )
)
speech_commands_v1["train"] = speech_commands_v1["train"].select(
    [i for i in range((len(speech_commands_v1["train"]) // BATCH_SIZE) * BATCH_SIZE)]
)
speech_commands_v1["test"] = speech_commands_v1["test"].select(
    [i for i in range((len(speech_commands_v1["test"]) // BATCH_SIZE) * BATCH_SIZE)]
)
print(speech_commands_v1)
```



```
DatasetDict({
    train: Dataset({
        features: ['file', 'audio', 'label'],
        num_rows: 15296
    })
    test: Dataset({
        features: ['file', 'audio', 'label'],
        num_rows: 15296
    })
})
```

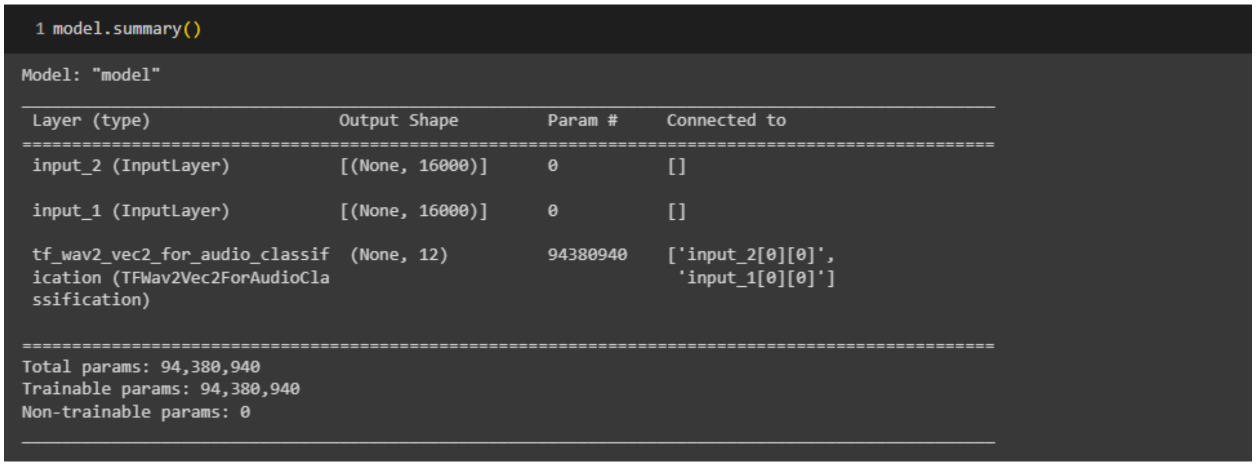
## Building and Compiling Model

Now  we will build and compile our model. We use the SparseCategoricalCrossentropy to train our model since it is a classification task. Following much literature we evaluate our model on the accuracy metric.



```python
def build_model():
    # Model's input
    inputs = {
        "input_values": tf.keras.Input(shape=(MAX_SEQ_LENGTH,), dtype="float32"),
        "attention_mask": tf.keras.Input(shape=(MAX_SEQ_LENGTH,), dtype="int32"),
    }
    # Instantiate the Wav2Vec 2.0 model with Classification-Head using the desired
    # pre-trained checkpoint
    wav2vec2_model = TFWav2Vec2ForAudioClassification(MODEL_CHECKPOINT, NUM_CLASSES)(
        inputs
    )
    # Model
    model = tf.keras.Model(inputs, wav2vec2_model)
    # Loss
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False)
    # Optimizer
    optimizer = keras.optimizers.legacy.Adam(learning_rate=1e-5)
    # Compile and return
    model.compile(loss=loss, optimizer=optimizer, metrics=["accuracy"])
    return model


model = build_model()
```

**Model Summary:** here you can see model summary

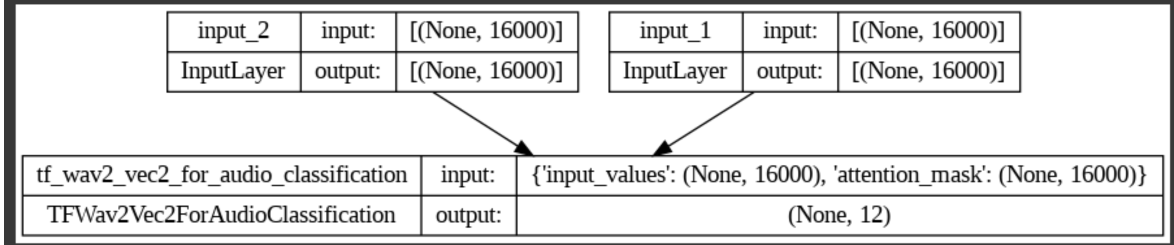


```
model.summary()
```

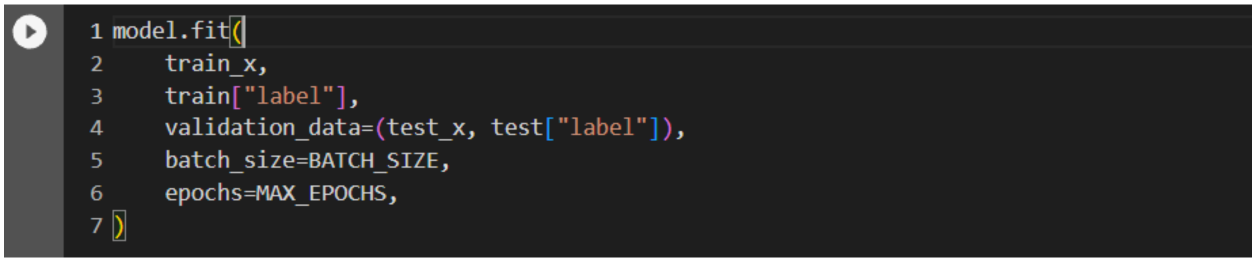
```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 16000)]      0           []

 input_1 (InputLayer)           [(None, 16000)]      0           []

 tf_wav2_vec2_for_audio_classif  (None, 12)          94380940    ['input_2[0][0]',
 ication (TFWav2Vec2ForAudioCla                                   'input_1[0][0]']
 ssification)

==================================================================================================
Total params: 94,380,940
Trainable params: 94,380,940
Non-trainable params: 0
__________________________________________________________________________________________________
```

here you can see model plot

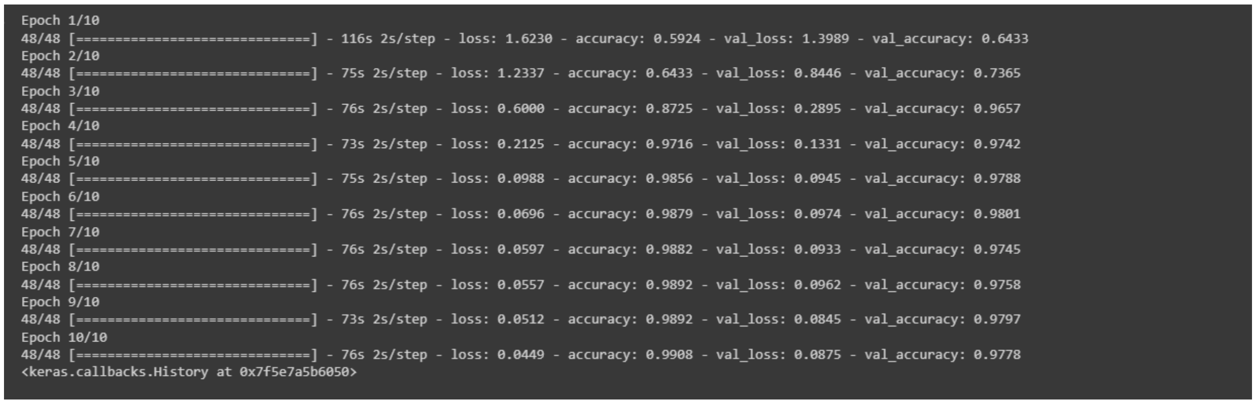
```
tf.keras.utils.plot_model(model,show_shapes=True)
```


input_2 | input: | [(None, 16000)]
InputLayer | output: | [(None, 16000)]
input_1 | input: | [(None, 16000)]
InputLayer | output: | [(None, 16000)]
tf_wav2_vec2_for_audio_classification | input: | {'input_values': (None, 16000), 'attention_mask': (None, 16000)}
TFWav2Vec2ForAudioClassification | output: | (None, 12)

**Training Model:** Now we will start training our model.



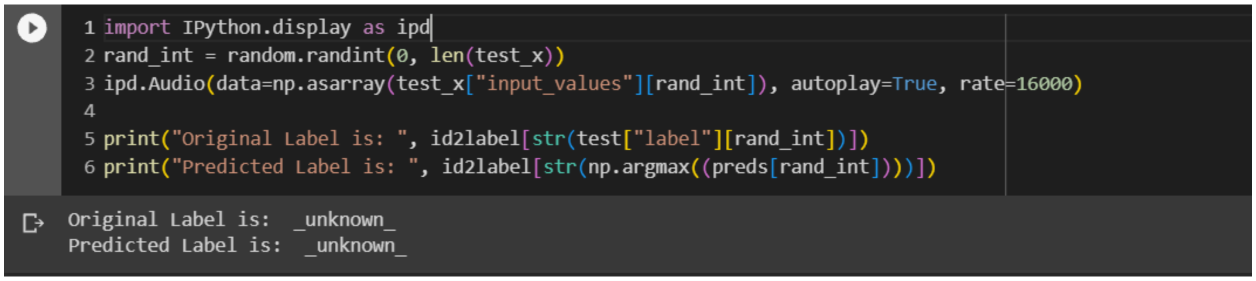
```
model.fit(
    train_x,
    train["label"],
    validation_data=(test_x, test["label"]),
    batch_size=BATCH_SIZE,
    epochs=MAX_EPOCHS,
)
```



```
Epoch 1/10
48/48 [==============================] - 116s 2s/step - loss: 1.6230 - accuracy: 0.5924 - val_loss: 1.3989 - val_accuracy: 0.6433
Epoch 2/10
48/48 [==============================] - 75s 2s/step - loss: 1.2337 - accuracy: 0.6433 - val_loss: 0.8446 - val_accuracy: 0.7365
Epoch 3/10
48/48 [==============================] - 76s 2s/step - loss: 0.6000 - accuracy: 0.8725 - val_loss: 0.2895 - val_accuracy: 0.9657
Epoch 4/10
48/48 [==============================] - 73s 2s/step - loss: 0.2125 - accuracy: 0.9716 - val_loss: 0.1331 - val_accuracy: 0.9742
Epoch 5/10
48/48 [==============================] - 75s 2s/step - loss: 0.0988 - accuracy: 0.9856 - val_loss: 0.0945 - val_accuracy: 0.9788
Epoch 6/10
48/48 [==============================] - 76s 2s/step - loss: 0.0696 - accuracy: 0.9879 - val_loss: 0.0974 - val_accuracy: 0.9801
Epoch 7/10
48/48 [==============================] - 76s 2s/step - loss: 0.0597 - accuracy: 0.9882 - val_loss: 0.0933 - val_accuracy: 0.9745
Epoch 8/10
48/48 [==============================] - 76s 2s/step - loss: 0.0557 - accuracy: 0.9892 - val_loss: 0.0962 - val_accuracy: 0.9758
Epoch 9/10
48/48 [==============================] - 73s 2s/step - loss: 0.0512 - accuracy: 0.9892 - val_loss: 0.0845 - val_accuracy: 0.9797
Epoch 10/10
48/48 [==============================] - 76s 2s/step - loss: 0.0449 - accuracy: 0.9908 - val_loss: 0.0875 - val_accuracy: 0.9778
<keras.callbacks.History at 0x7f5e7a5b6050>
```

**Testing Model**

Great! After our model has been trained, we use it to predict the classes for the audio samples in the test set using model.predict() method.



```
import IPython.display as ipd
rand_int = random.randint(0, len(test_x))
ipd.Audio(data=np.asarray(test_x["input_values"][rand_int]), autoplay=True, rate=16000)

print("Original Label is: ", id2label[str(test["label"][rand_int])])
print("Predicted Label is: ", id2label[str(np.argmax((preds[rand_int])))])
```

```
Original Label is:  _unknown_
Predicted Label is:  _unknown_
```

**Accuracy: 99.08%**

Here you can find full code Audio-classification-model.

## Conclusion:

In this article, we have explored the implementation of transformer models for audio data analysis. By preprocessing the audio data, preparing a dataset, defining the transformer model, and training and evaluating it, we have demonstrated the power of transformers in audio analysis tasks. The code examples and output showcase how transformers can be leveraged for sound classification. With further experimentation and optimization, you can unlock the full potential of transformers in audio data analysis.

If you want to read more articles than you can refer to aiensured blog.

## References:

1. wikipedia

2. https://keras.io/examples/audio/wav2vec2_audiocls/

3. https://github.com/SinghManish1/music_genre_classification

4. https://medium.com/tensorflow/a-transformer-chatbot-tutorial-with-tensorflow-2-0-88bf59e66fe2

5. Attention Is All You Need

6. https://www.kaggle.com/maxwell110/beginner-s-guide-to-audio-data-2

7. https://kikaben.com/transformers-encoder-decoder/