January 12, 2023

Contents

1 Introduction

Speech Emotion Recognition (SER) is a system that can identify an emotion range or sentimental value in various audio recordings. It has several fields of application such as job interviews, caller-agent calls or streaming videos.
To perform this task, we will use a network trained from scratch with 1 CNN, 2 CNNs in parallel and then 2 CNNs and a transformer encoder block, these 3 in parallel. Then we will use two pre-trained networks : VGG16 with feature extraction and fine tuning and Wav2Vec2 with fine tuning.

2 Related Works

The task of emotion recognition from speech can be solved by many different methods. The authors in [1] proposed a method based on concatenated convolution neural networks (CNNs) and recurrent neural networks (RNNs) without using any traditional hand-crafted features.
Other methods containing long short-term memory neural networks (LSTMs) are also used. The authors in [2] proposed a system that combines a deep convolutional neural network (DCNN) and a bidirectional long-short term memory (BLSTM) network. They managed to reach 82.7 % of accuracy on the RAVDESS dataset.
The authors in [3] proposed architectures such as CNNs and LSTM to test the emotion capturing capability from various standard speech representations such as mel spectrogram, magnitude spectrogram and Mel-Frequency Cepstral Coefficients (MFCC's).
The authors in [4] proposed a deep autoencoder based on a Multilayer perceptron.

All these papers give us several tracks to exploit about data pre-processing and information to extract and about some networks to use.

3 Dataset

In this project we employ the following datasets:

- Ryerson Audio-Visual Database of Emotional Speech and Song. (RAVDESS)
- Surrey Audio-Visual Expressed Emotion. (SAVEE)
- Toronto emotional speech set. (TESS)
- Crowd-sourced Emotional Multimodal Actors Dataset. (CREMA-D)

The datasets were recorded from English speaker actors and hand labeled by different people who took part in each project. We are going to use the audio-only speech portion of the previous datasets.

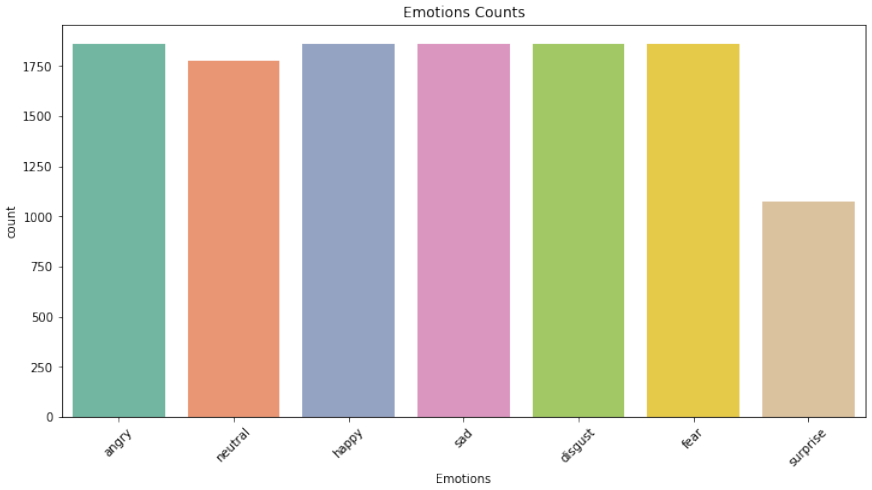


Figure 1: class distribution.

4 Models from scratch

4.1 Feature Extraction

Feature extraction involves the selection and extraction of relevant features from raw audio data that can effectively represent the emotions conveyed in the speech. These features can then be fed into a machine learning model for training and emotion prediction. There are several feature extraction techniques that have been proposed and used in the literature for speech emotion recognition. We will use two of them: *Mel Spectrograms* and *mel-frequency cepstral coefficients* (MFCCs).

The feature extraction processed is carried out using the Python *Librosa* library: it provides a wide range of functionality for working with audio data, including functions for loading audio files, feature extraction, and visualization.

We extracted waveforms from the audio data with a sample rate of 16000 due to the limited RAM that Google Colab provides.

4.1.1 Mel Spectrogram

A Mel spectrogram is a visual representation of the spectral characteristics of a speech signal, with the x-axis representing time, the y-axis representing frequency, and the intensity of each point representing the power or magnitude of the signal at that time and frequency. It is created by first applying a short-time Fourier transform (STFT) to the speech signal, which decomposes the signal into its frequency components over time, that are then mapped to the Mel scale using a non-linear transformation. The *Mel scale* is a scale of frequencies that is more closely aligned with the perceived pitch of human speech.

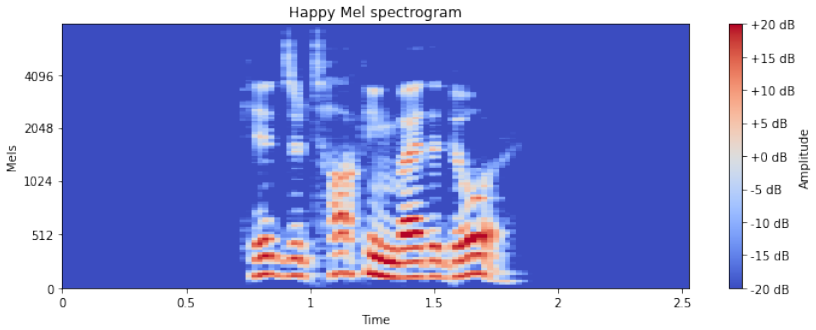


Figure 2: Mel Spectrogram of an Happy audio sample.

4.1.2 MFCC Coefficients

One popular technique is the use of Mel-Frequency Cepstral Coefficients (MFCCs). MFCCs are derived from the short-time Fourier transform (STFT) of the speech

signal and capture the spectral characteristics of the speech. They have been widely used in speech processing tasks such as speech recognition and emotion recognition due to their robustness to variations in speaking style and background noise.

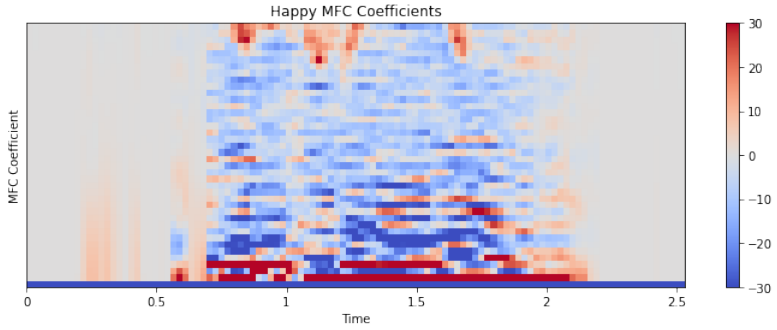


Figure 3: MFCCs of of an Happy audio sample.

4.2 Data Augumentation Techniques

Although the dataset has a reasonably high number of samples, it can be useful to employ data augmentation techniques to create pseudo-new training samples: some of the datasets that we employed, e.g. RAVDESS, have extremely clean audio while in real situations it is typically not so since real world data are likely to be noisy; By performing data augmentation we can make our models more robust to noise by creating new trained samples by simply adding adding noise to the original ones.

4.2.1 Additive White Gaussian Noise (AWGN)

Additive white Gaussian noise (AWGN) is a basic noise model used in information theory to mimic the effects of various sources of noise that occur in nature and that can degrade the quality of the signal.

When a signal is corrupted by AWGN, a gaussian noise vector sampled from a normal distribution with a mean of zero and a certain variance and is added uniformly across the frequency distribution as an independent and identically distributed random variable. The variance of the noise is a measure of the amount of noise added to the signal and is usually specified in terms of the signal-to-noise ratio (SNR) of the system. The SNR is defined as the ratio of the power of the signal to the power of the noise, and is usually given in decibels (dB).

A comparison of the original audio signal's waveform and mel-spectrogram compared with the AWGN augmented signal is shown on Figure 4.

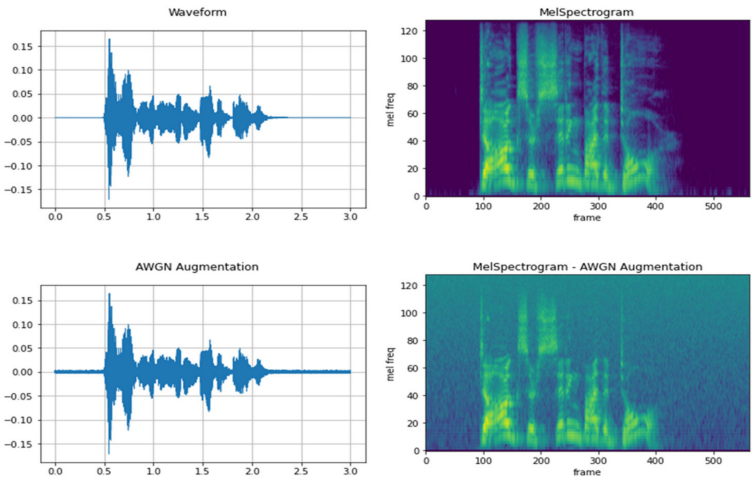


Figure 4: Comparison of the original audio signal's waveform and mel-spectrogram compared with an Additive White Gaussian Noise augmented signal [5].

4.3 First Model: Single CNN

For this network and for the following one we will consider only MFCCs as extracted features, we will go through the results with Mel Spectrograms as extracted features only in the final network.

4.3.1 Convolutional Neural Network Block

The architecture of the CNN that will be used in this and in the following sections consists of a 3 layer CNN, with increasing feature maps, inspired by the 2D convolutional blocks of the LeNet architecture (Conv→ Pool→Conv→Pool→FC); the idea of increasing the complexity of feature maps at each CNN layes is taken up by the AlexNet architecture.

The convolution block takes an input with the format of batch size, channel, height, and width: the input feature with MFCC features has a shape of (N, 1, 128, 94), where the N is the number of training data. Note that an MFCC diagram can be considered as a grey-scale image so we use a single channel rather than the usual 3 of RGB images. The channel consists of the intensities for the 128 MFCC bands at 94 timesteps.

The first layer only has a single input channel, and so a 1x3x3 filter that produces an output of 16 channels, with a batch normalization applied to the output feature map before using an ReLU activation function. A 2x2 with stride 2 max-pooling layer, is applied after the activation, followed by a drop-out layer with the probability of 0.3 on all subsequent layers. The second layer expands the output feature map (16 input channels) to a depth of 32 channels, so creating a 16x3x3 filter, and it increases the the max pool kernel size to 4x4 with stride 4.

The last layer has 32 input channels, so a 32x3x3 filter, and 64 output channels, using the same max pool kernel of the second layer.

The architecture is given below (the summary and the diagram).

Model: "model"

Layer (**type**)	Output Shape	Param #
input_2 (InputLayer)	[(None, 1, 128, 94)]	0
conv2d_3 (Conv2D)	(None, 16, 128, 94)	160
batch_normalization_3 (BatchNormalization)	(None, 16, 128, 94)	376
activation_3 (Activation)	(None, 16, 128, 94)	0
max_pooling2d_3 (MaxPooling2D)	(None, 16, 64, 47)	0
dropout_3 (Dropout)	(None, 16, 64, 47)	0
conv2d_4 (Conv2D)	(None, 32, 64, 47)	4640
batch_normalization_4 (BatchNormalization)	(None, 32, 64, 47)	188
activation_4 (Activation)	(None, 32, 64, 47)	0
max_pooling2d_4 (MaxPooling2D)	(None, 32, 16, 11)	0
dropout_4 (Dropout)	(None, 32, 16, 11)	0
conv2d_5 (Conv2D)	(None, 64, 16, 11)	18496
batch_normalization_5 (BatchNormalization)	(None, 64, 16, 11)	44
activation_5 (Activation)	(None, 64, 16, 11)	0
max_pooling2d_5 (MaxPooling2D)	(None, 64, 4, 2)	0
dropout_5 (Dropout)	(None, 64, 4, 2)	0
first_block (Flatten)	(None, 512)	0
dense (Dense)	(None, 7)	3591
softmax (Softmax)	(None, 7)	0

Total params: 27,495
Trainable params: 27,191
Non−trainable params: 304

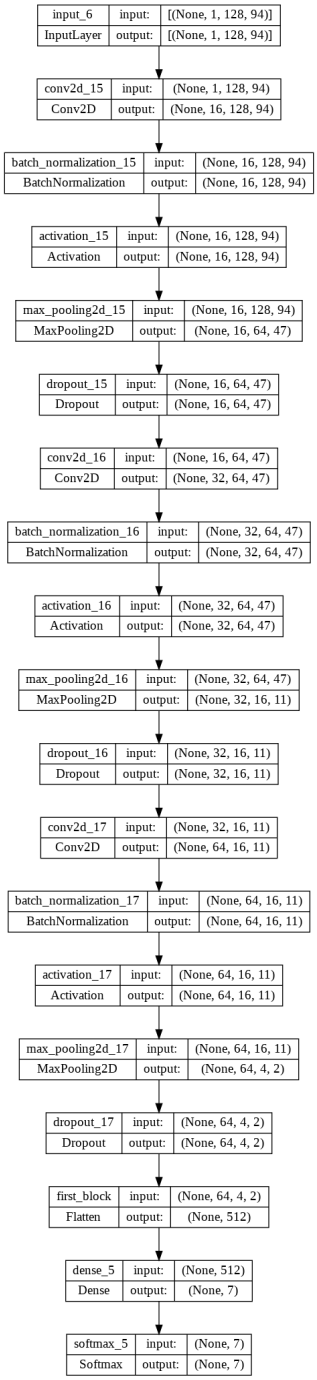


Figure 5: CNN Model Architecture for MFCC input features.

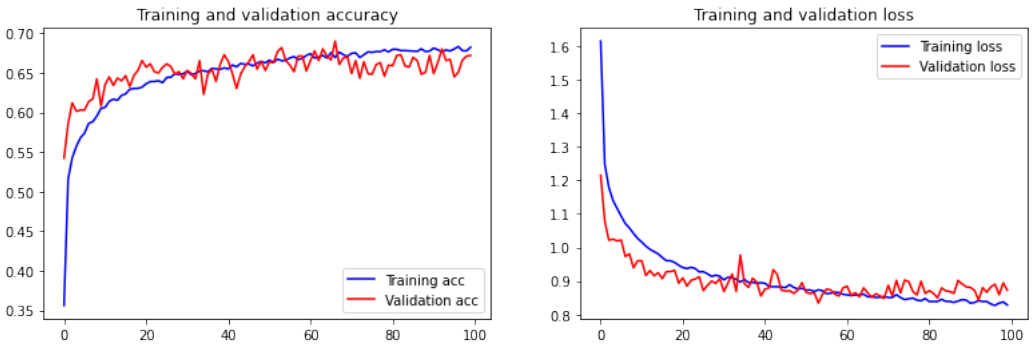
4.3.2 Training of the network

The first network from scratch is done by using a single CNN block. We used the Keras Tuner, in detail the HyperBand tuner, to find the best hyperparameters. We selected different values of *learning_rate* building an hypermodel defining the model architecture and the hyperparameter search space. We can see in the table below the results provided by Tensorboard, and the best validation accuracy is obtained using a learning rate equal to 1e-2.

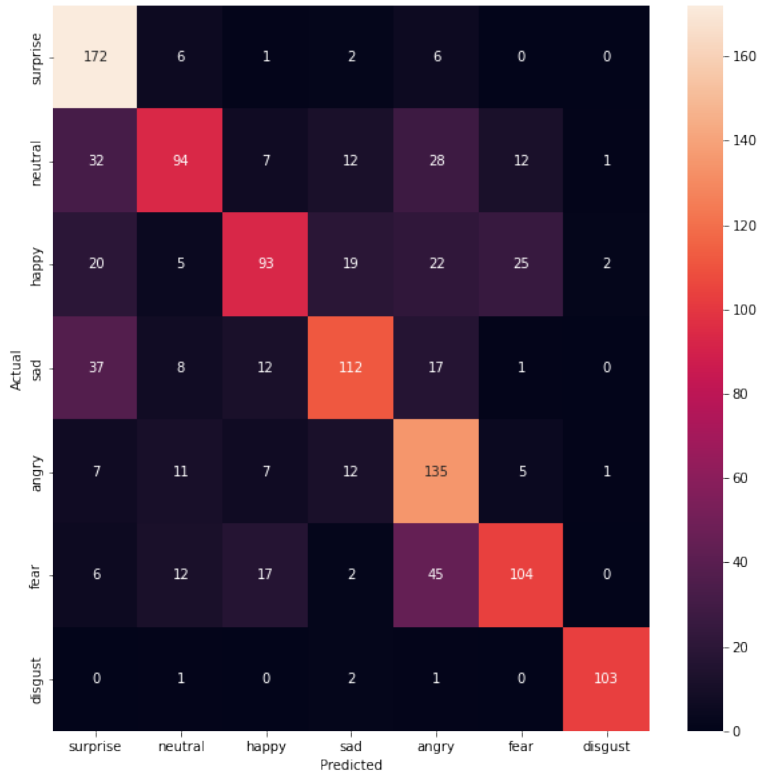
learning_rate	t_epoch_accuracy	v_epoch_accuracy	t_epoch_loss	v_epoch_loss
$1 \cdot 10^{-5}$	0.152	0.178	2.61	1.96
0.0001	0.218	0.163	1.89	1.88
0.001	0.437	0.496	1.42	1.28
0.01	0.481	0.565	1.32	1.16

4.3.3 Results

In all the evaluations the dataset is splitted in 80% train 10% validation and 10% test. The data augmentation process is performed only on the training data, the Gaussian white noise is samples with random SNR values in the range [15,30]. The model has been optimized using Adam optimizer with the tuned 1e-2 learning rate, and the *categorical crossentropy* is used as loss function. The model has been trained with a maximum number of epochs of 100, and after the first run we can see from the graph below that the best number of epoch for this model is equal to 67, so we retrain the model with the correct number of epochs and evaluate its performance that are show in the following table.



	Precision	Recall	F1 Score	Support
Angry	0.63	0.92	0.75	187
Disgust	0.69	0.51	0.58	186
Fear	0.68	0.50	0.58	186
Happy	0.70	0.60	0.64	187
Neutral	0.53	0.76	0.62	178
Sad	0.71	0.56	0.62	186
Surprise	0.96	0.96	0.96	107
Accuracy			0.67	1217
Macro AVG	0.70	0.69	0.68	1217
Weighted AVG	0.68	0.67	0.66	1217



4.4 Second Model: Two CNNs in Parallel

The second network architecture consists in parallelizing two of the CNN blocks that have been presented in the previous section. Inception and GoogLeNet architecture are the inspiration for parallelizing CNN layers in the hopes of diverisfying the features learned by the network. The architecture is given below (the summary and the diagram).

Model: "model"

Layer (**type**)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 1, 128, 94)]	0	[]
conv2d (Conv2D)	(None, 16, 128, 94)	160	['input_1 [0][0] ']
conv2d_3 (Conv2D)	(None, 16, 128, 94)	160	['input_1 [0][0] ']
batch_normalization (BatchNormaliz ation)	(None, 16, 128, 94)	376	['conv2d [0][0] ']
batch_normalization_3 (BatchNormal ization)	(None, 16, 128, 94)	376	['conv2d_3 [0][0] ']
activation (Activation)	(None, 16, 128, 94)	0	['batch_normalization [0][0] ']
activation_3 (Activation)	(None, 16, 128, 94)	0	['batch_normalization_3 [0][0] ']
max_pooling2d (MaxPooling2D)	(None, 16, 64, 47)	0	['activation [0][0] ']
max_pooling2d_3 (MaxPooling2D)	(None, 16, 64, 47)	0	['activation_3 [0][0] ']
dropout (Dropout)	(None, 16, 64, 47)	0	['max_pooling2d[0][0]']
dropout_3 (Dropout)	(None, 16, 64, 47)	0	['max_pooling2d_3[0][0]']
conv2d_1 (Conv2D)	(None, 32, 64, 47)	4640	['dropout [0][0] ']
conv2d_4 (Conv2D)	(None, 32, 64, 47)	4640	['dropout_3 [0][0] ']
batch_normalization_1 (BatchNormal ization)	(None, 32, 64, 47)	188	['conv2d_1 [0][0] ']
batch_normalization_4 (BatchNormal ization)	(None, 32, 64, 47)	188	['conv2d_4 [0][0] ']
activation_1 (Activation)	(None, 32, 64, 47)	0	['batch_normalization_1 [0][0] ']
activation_4 (Activation)	(None, 32, 64, 47)	0	['batch_normalization_4 [0][0] ']
max_pooling2d_1 (MaxPooling2D)	(None, 32, 16, 11)	0	['activation_1 [0][0] ']
max_pooling2d_4 (MaxPooling2D)	(None, 32, 16, 11)	0	['activation_4 [0][0] ']
dropout_1 (Dropout)	(None, 32, 16, 11)	0	['max_pooling2d_1[0][0]']
dropout_4 (Dropout)	(None, 32, 16, 11)	0	['max_pooling2d_4[0][0]']
conv2d_2 (Conv2D)	(None, 64, 16, 11)	18496	['dropout_1 [0][0] ']
conv2d_5 (Conv2D)	(None, 64, 16, 11)	18496	['dropout_4 [0][0] ']
batch_normalization_2 (BatchNormal ization)	(None, 64, 16, 11)	44	['conv2d_2 [0][0] ']
batch_normalization_5 (BatchNormal ization)	(None, 64, 16, 11)	44	['conv2d_5 [0][0] ']
activation_2 (Activation)	(None, 64, 16, 11)	0	['batch_normalization_2 [0][0] ']
activation_5 (Activation)	(None, 64, 16, 11)	0	['batch_normalization_5 [0][0] ']
max_pooling2d_2 (MaxPooling2D)	(None, 64, 4, 2)	0	['activation_2 [0][0] ']
max_pooling2d_5 (MaxPooling2D)	(None, 64, 4, 2)	0	['activation_5 [0][0] ']
dropout_2 (Dropout)	(None, 64, 4, 2)	0	['max_pooling2d_2[0][0]']
dropout_5 (Dropout)	(None, 64, 4, 2)	0	['max_pooling2d_5[0][0]']
first_block (Flatten)	(None, 512)	0	['dropout_2 [0][0] ']
second_block (Flatten)	(None, 512)	0	['dropout_5 [0][0] ']
tf.concat (TFOpLambda)	(None, 1024)	0	[' first_block [0][0] ',

```
                                                                  'second_block[0][0]']
dense (Dense)                (None, 7)                 7175       ['tf.concat[0][0]']
softmax (Softmax)            (None, 7)                 0          ['dense[0][0]']
==================================================================================================
Total params: 54,983
Trainable params: 54,375
Non-trainable params: 608
__________________________________________________________________________________________________
```

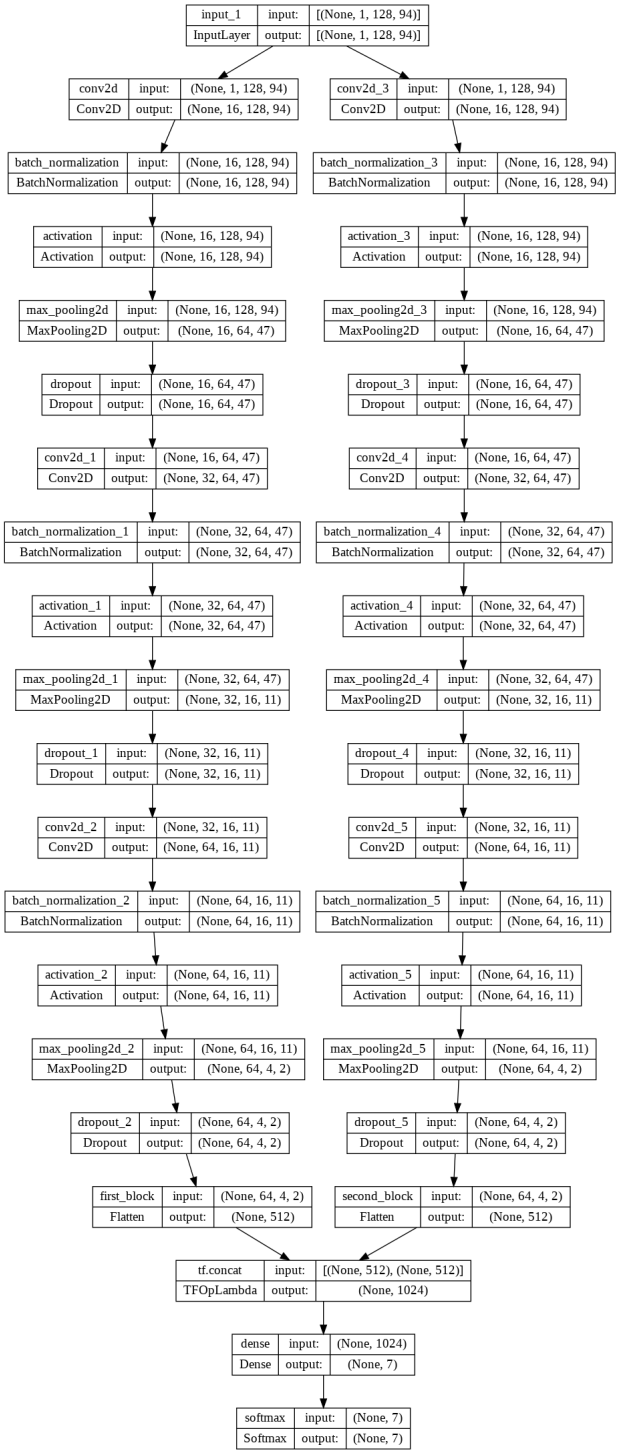


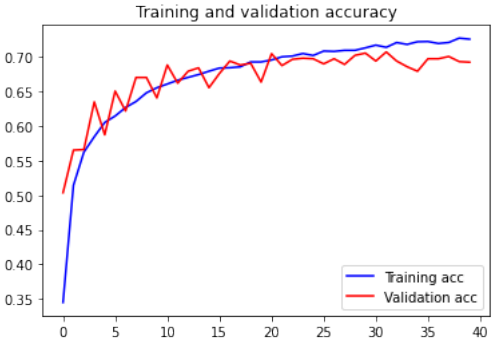
Figure 6: Two CNNs Model Architecture for MFCC input features.

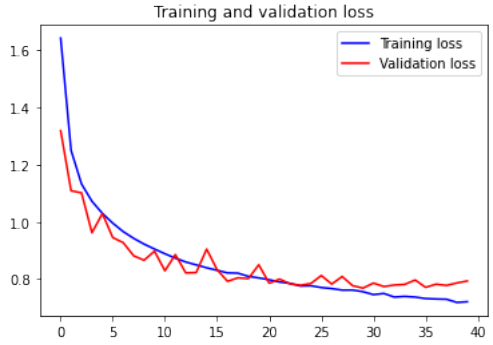
4.4.1 Results

In this section will be analyzed the results obtained training the Two CNNs Model with 2-fold AWGN augmentation and with MFCCs as extracted features.

In all the evaluations the dataset is splitted in 80% train 10% validation and 10% test. The data augmentation process is performed only on the training data, the Gaussian white noise is samples with random SNR values in the range [15,30]. All the models have been optimized using Adam optimizer with 1e-4 learning rate, and the *categorical crossentropy* is used as loss function. The models have been trained with a maximum number of epochs of 100 and with early stopping monitoring the validation loss and a patience value of 10.

	Precision	Recall	F1 Score	Support
Angry	0.78	0.84	0.81	187
Disgust	0.69	0.58	0.63	186
Fear	0.72	0.56	0.63	186
Happy	0.62	0.77	0.68	187
Neutral	0.61	0.72	0.66	178
Sad	0.69	0.60	0.64	186
Surprise	0.96	0.96	0.96	107
Accuracy			0.70	1217
Macro AVG	0.72	0.72	0.72	1217
Weighted AVG	0.71	0.70	0.70	1217





4.5 Third Model: Two CNNs and a Transformer in Parallel

The model architecture is inspired by the one proposed by authors in [5]: two CNN blocks, with 3 layers each, are implemented in parallel with a Transformer Encoder block, the output of the three blocks are then concatenated and processed by a feed forward network for the final prediction. The architecture of each CNN block is the same proposed in the previous sections.

4.5.1 Transformer Encoder Block

The Transformer architecture employed in the Transformer Encoder block of the network is the one proposed by authors in [6], the only difference is that the encoder that we employed is composed by 4 stacked encoders (each with self-attention and feed forward layers) instead of the 6 proposed by the original paper.

The goal of using a Transformer-Encoder layer is to help predict the emotion based on the global structure of the feature diagrams of each emotion: the multi-head self-attention layer of the transformer allows the network to take into consideration all the previous time steps when predicting the next.

The Transformer-Encoder provided by *keras_nlp* follows the architecture of the transformer encoder layer of the original paper, users can instantiate multiple instances of this class to stack up an encoder: the full transformer encoder is obtained by stacking 4 identical encoder layers, each of which has 4 attention heads, a feed forward network with 512 neurons and ReLU activation function; the dropout value of both the FF and the Encoders is 0.4. The output of the final encoder is then averaged with respect to the time step (e.g. dim 128x94 $->$ 128).

Since the *keras_nlp* Transformer Encoder requires input tensor in the format of timestep, batch, embedding (freq), some preprocessing is needed to transform the input of the overall network in the correct form: firstly, the channel dimension is removed from the input tensor and then the tensor dimension are reordered following the required format.

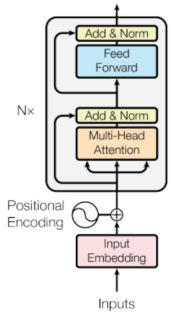


Figure 7: Transformer Encoder Architecture proposed in [6].

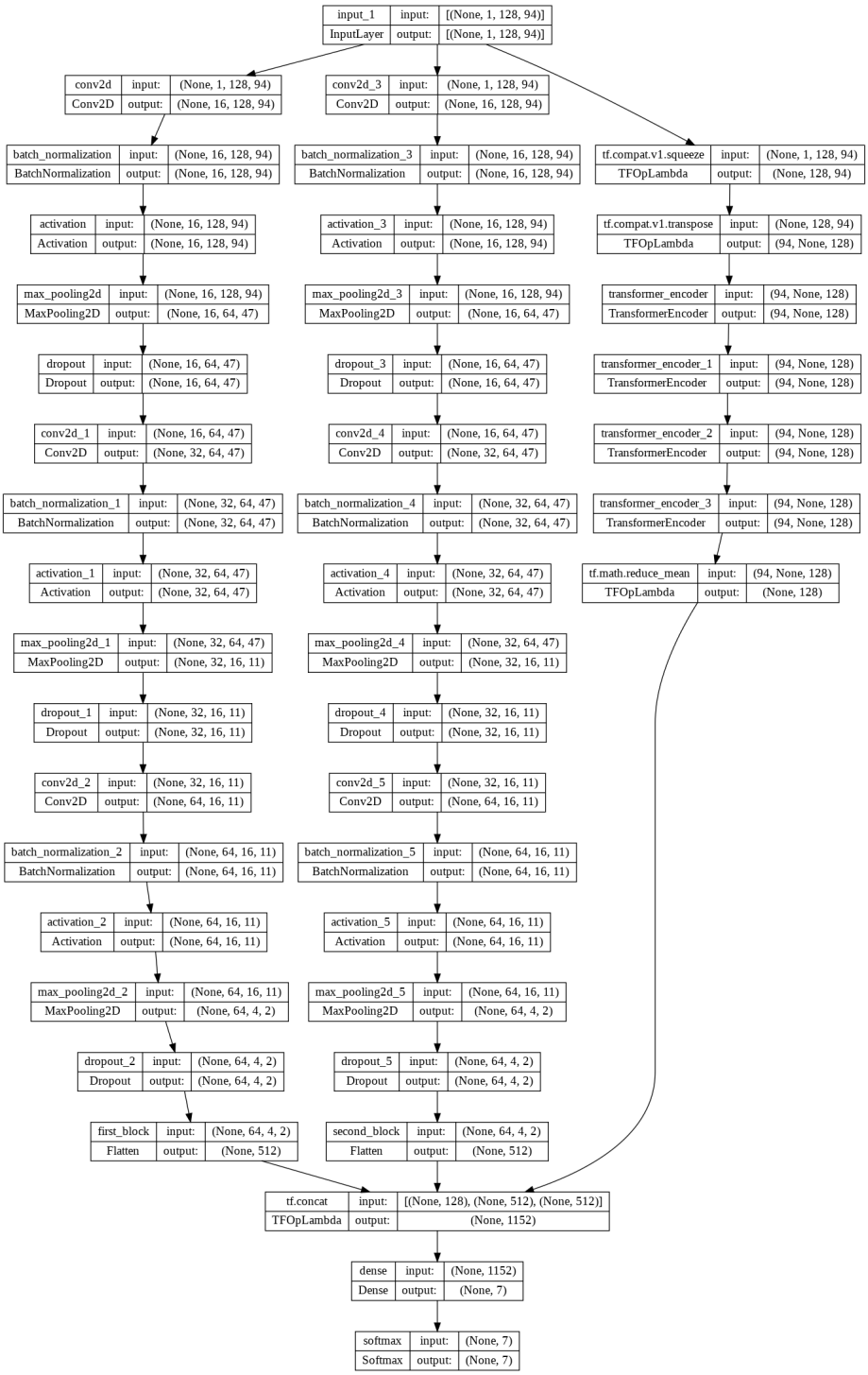


Figure 8: Parallel Model Architecture for MFCC input features.

Model: "ParallelModel_MFCC"

Layer (**type**)	Output Shape	Param #	Connected to
input_5 (InputLayer)	[(None, 1, 128, 94)]	0	[]
conv2d_24 (Conv2D)	(None, 16, 128, 94)	160	['input_5 [0][0] ']
conv2d_27 (Conv2D)	(None, 16, 128, 94)	160	['input_5 [0][0] ']
batch_norm_24 (BatchNorm)	(None, 16, 128, 94)	376	['conv2d_24[0][0] ']
batch_norm_27 (BatchNorm)	(None, 16, 128, 94)	376	['conv2d_27[0][0] ']
activation_24 (Activation)	(None, 16, 128, 94)	0	['batch_normalization_24[0][0] ']
activation_27 (Activation)	(None, 16, 128, 94)	0	['batch_normalization_27[0][0] ']
max_pooling2d_24 (MaxPooling2D)	(None, 16, 64, 47)	0	['activation_24 [0][0] ']
max_pooling2d_27 (MaxPooling2D)	(None, 16, 64, 47)	0	['activation_27 [0][0] ']
dropout_24 (Dropout)	(None, 16, 64, 47)	0	['max_pooling2d_24[0][0]']
dropout_27 (Dropout)	(None, 16, 64, 47)	0	['max_pooling2d_27[0][0]']
conv2d_25 (Conv2D)	(None, 32, 64, 47)	4640	['dropout_24[0][0] ']
conv2d_28 (Conv2D)	(None, 32, 64, 47)	4640	['dropout_27[0][0] ']
batch_norm_25 (BatchNorm)	(None, 32, 64, 47)	188	['conv2d_25[0][0] ']
batch_norm_28 (BatchNorm)	(None, 32, 64, 47)	188	['conv2d_28[0][0] ']
activation_25 (Activation)	(None, 32, 64, 47)	0	['batch_normalization_25[0][0] ']
activation_28 (Activation)	(None, 32, 64, 47)	0	['batch_normalization_28[0][0] ']
max_pooling2d_25 (MaxPooling2D)	(None, 32, 16, 11)	0	['activation_25 [0][0] ']
max_pooling2d_28 (MaxPooling2D)	(None, 32, 16, 11)	0	['activation_28 [0][0] ']
tf.compat.v1.squeeze_4 (TFOpLambda)	(None, 128, 94)	0	['input_5 [0][0] ']
dropout_25 (Dropout)	(None, 32, 16, 11)	0	['max_pooling2d_25[0][0]']
dropout_28 (Dropout)	(None, 32, 16, 11)	0	['max_pooling2d_28[0][0]']
tf.compat.v1.transpose_4 (TFOpLambda)	(94, None, 128)	0	['tf.compat.v1.squeeze_4[0][0] ']
conv2d_26 (Conv2D)	(None, 64, 16, 11)	18496	['dropout_25[0][0] ']
conv2d_29 (Conv2D)	(None, 64, 16, 11)	18496	['dropout_28[0][0] ']
transformer_encoder_16 (Encoder)	(94, None, 128)	198272	['tf.compat.v1.transpose_4[0][0] ']
batch_norm_26 (BatchNorm)	(None, 64, 16, 11)	44	['conv2d_26[0][0] ']
batch_norm_29 (BatchNorm)	(None, 64, 16, 11)	44	['conv2d_29[0][0] ']
transformer_encoder_17 (Encoder)	(94, None, 128)	198272	['transformer_encoder_16[0][0] ']
activation_26 (Activation)	(None, 64, 16, 11)	0	['batch_normalization_26[0][0] ']
activation_29 (Activation)	(None, 64, 16, 11)	0	['batch_normalization_29[0][0] ']
transformer_encoder_18 (Encoder)	(94, None, 128)	198272	['transformer_encoder_17[0][0] ']
max_pooling2d_26 (MaxPooling2D)	(None, 64, 4, 2)	0	['activation_26 [0][0] ']
max_pooling2d_29 (MaxPooling2D)	(None, 64, 4, 2)	0	['activation_29 [0][0] ']
transformer_encoder_19 (Encoder)	(94, None, 128)	198272	['transformer_encoder_18[0][0] ']
dropout_26 (Dropout)	(None, 64, 4, 2)	0	['max_pooling2d_26[0][0]']
dropout_29 (Dropout)	(None, 64, 4, 2)	0	['max_pooling2d_29[0][0]']
tf.math.reduce_mean_4 (TFOpLambda)	(None, 128)	0	['transformer_encoder_19[0][0] ']
first_block (Flatten)	(None, 512)	0	['dropout_26[0][0] ']
second_block (Flatten)	(None, 512)	0	['dropout_29[0][0] ']
tf.concat_4 (TFOpLambda)	(None, 1152)	0	['tf.math.reduce_mean_4[0][0]', ' first_block [0][0] ', 'second_block[0][0] ']
dense_4 (Dense)	(None, 7)	8071	['tf.concat_4 [0][0] ']
softmax_4 (Softmax)	(None, 7)	0	['dense_4 [0][0] ']

Total params: 848,967
Trainable params: 848,359
Non−trainable params: 608

4.5.2 Results

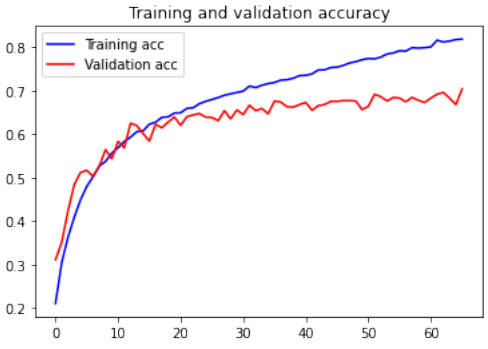
In this section will be analyzed the results obtained training the Parallel Model with or without AWGN augmentation and with the feature extraction techniques that have been analyzed in the previous sections.

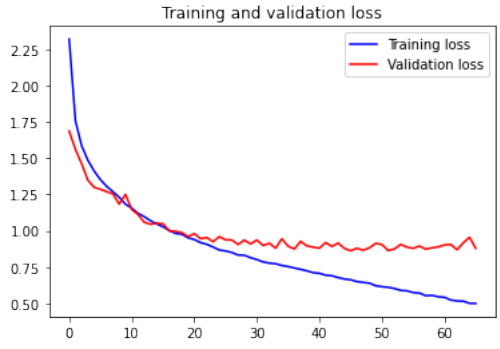
In all the evaluations the dataset is splitted in 80% train 10% validation and 10% test. The data augmentation process is performed only on the training data, the Gaussian white noise is samples with random SNR values in the range [15,30]. All the models have been optimized using Adam optimizer with 1e-4 learning rate, and the *categorical crossentropy* is used as loss function. The models have been trained with a maximum number of epochs of 100 and with early stopping monitoring the validation loss; different *patience* values have been used depending whether or not augmentation has been performed on the training data.

MFCCs w/o Data Augmentation

The first test was carried out extracting features as 128 MFCCs with an early stopping patience value of 20 epochs. Although we have not performed data augmentation, the results are slightly better than the one obtained with the two CNN model, that used augmented data.

	Precision	Recall	F1 Score	Support
Angry	0.81	0.89	0.85	187
Disgust	0.65	0.66	0.65	186
Fear	0.71	0.53	0.61	186
Happy	0.70	0.64	0.67	187
Neutral	0.64	0.65	0.64	178
Sad	0.63	0.73	0.68	186
Surprise	0.88	0.99	0.93	107
Accuracy			0.71	1217
Macro AVG	0.72	0.73	0.72	1217
Weighted AVG	0.71	0.71	0.71	1217



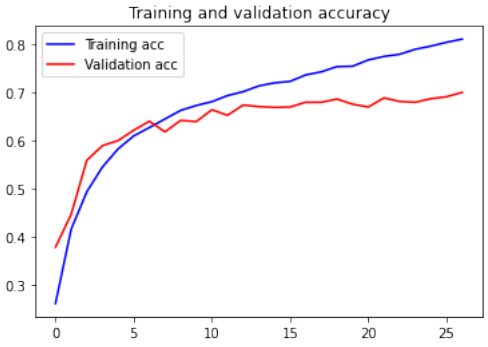


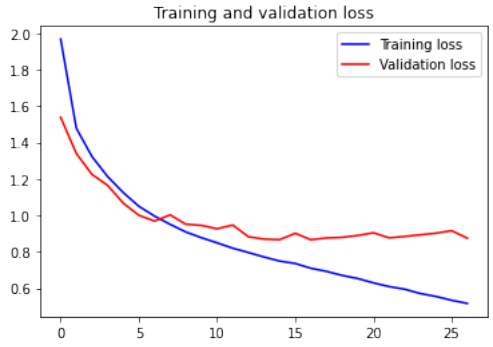
MFCCs w/ Data Augmentation

The second test is carried out using the same features as the previous one and with an higher number of training data thanks to a 2-fold data augmentation with AWGN. The total number of training samples is 29184. Compared to the previous test, the early stopping patience this time is set to 10 epochs due to the early overfitting caused by the higher quantity of training data.

Looking at the results we can see a clear improvement in the model: the F1 score values of each target class has improved compared to the values of the previous experiment, so augmenting the data has proved to be an effective technique to increase the performance.

	Precision	Recall	F1 Score	Support
Angry	0.92	0.81	0.86	187
Disgust	0.71	0.68	0.69	186
Fear	0.66	0.65	0.65	186
Happy	0.77	0.65	0.70	187
Neutral	0.65	0.66	0.66	178
Sad	0.61	0.77	0.68	186
Surprise	0.91	0.99	0.95	107
Accuracy			0.73	1217
Macro AVG	0.75	0.74	0.74	1217
Weighted AVG	0.74	0.73	0.73	1217



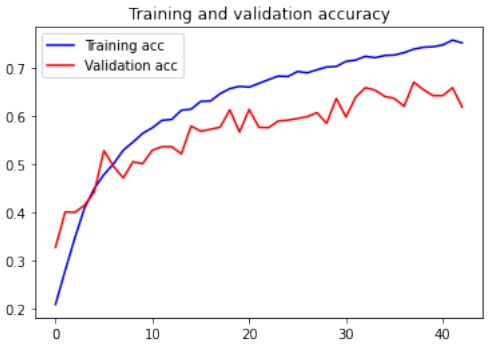


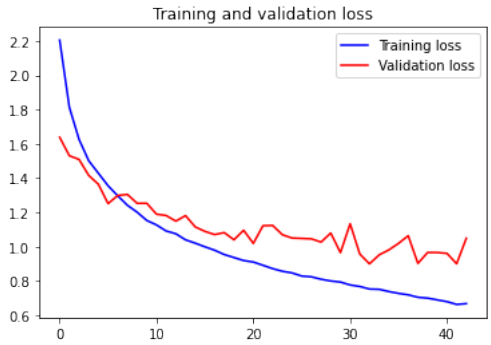
Mel Spectrograms w/o Data Augmentation

The third test is carried out using the other feature extraction technique that we presented: Mel Spectrograms. The input feature with MFCC features has a shape of (N, 1, 128, 188), where the N is the number of training data. As well as MFCC diagrams, also Mel Spectrograms can be considered as a grey-scale image so we can use once again a single channel rather than the usual 3 of RGB images.

Note that we have 188 timesteps rather than the 94 timesteps that we had with MFCCs features. Since we obtained worst results with 94 time step we increased the time step to 188 hoping to improve the model. In this report are shown only the results with 188 timesteps because they are actually very similar to the results with 94 timesteps. The early stopping patience value is set to 20 epochs.

	Precision	Recall	F1 Score	Support
Angry	0.86	0.61	0.72	187
Disgust	0.76	0.50	0.60	186
Fear	0.44	0.72	0.55	186
Happy	0.76	0.45	0.57	187
Neutral	0.59	0.69	0.64	178
Sad	0.57	0.73	0.64	186
Surprise	0.99	0.93	0.96	107
Accuracy			0.64	1217
Macro AVG	0.71	0.66	0.67	1217
Weighted AVG	0.69	0.64	0.65	1217

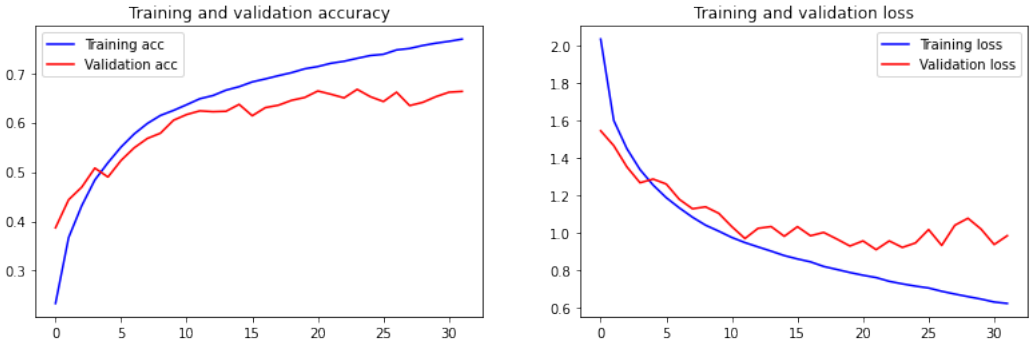




Mel Spectrograms w/ Data Augmentation

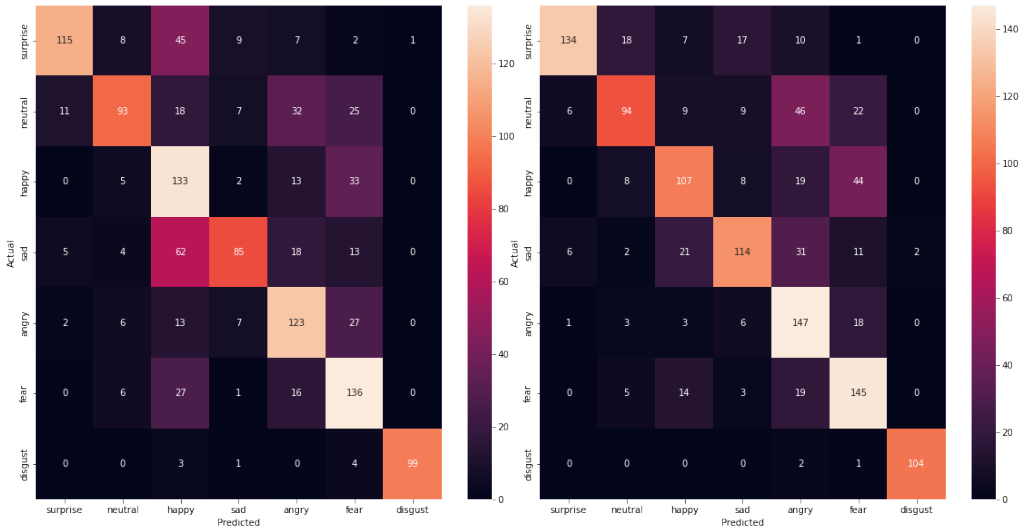
The forth and last test is carried out using the same features as the previous one and with an higher number of training data thanks to a 2-fold data augmentation with AWGN. The early stopping patience value is set to 10 epochs as it was for the model with augmented data and MFCCs features. Although we are still far from the results obtained with MFCCs, once again augmenting the data has proved to be an effective technique to increase the performance of the model.

	Precision	Recall	F1 Score	Support
Angry	0.91	0.72	0.80	187
Disgust	0.72	0.51	0.59	186
Fear	0.66	0.58	0.62	186
Happy	0.73	0.61	0.66	187
Neutral	0.54	0.83	0.65	178
Sad	0.60	0.78	0.68	186
Surprise	0.98	0.97	0.98	107
Accuracy			0.69	1217
Macro AVG	0.73	0.71	0.71	1217
Weighted AVG	0.72	0.69	0.69	1217

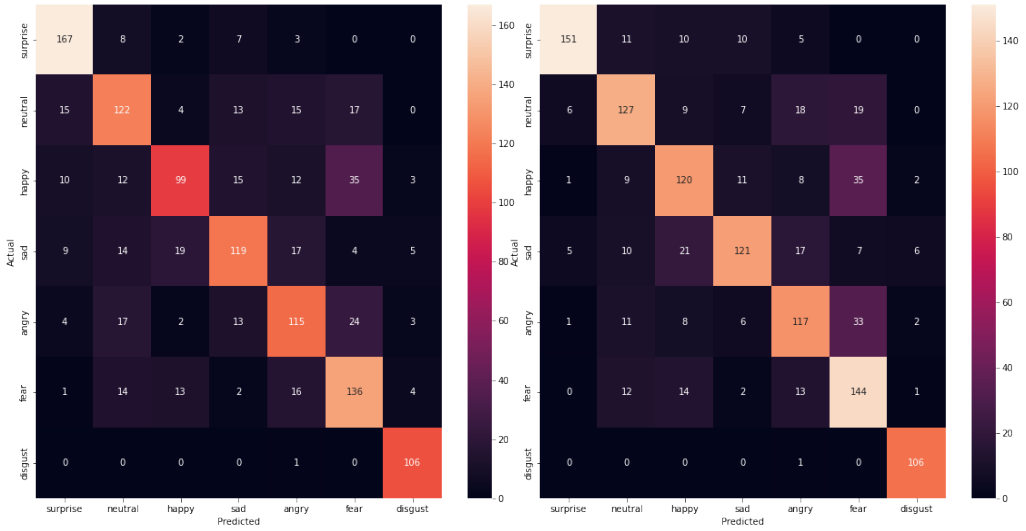


4.6 Final considerations

The best model has been the one with MFCC as extracted features and with 2-fold AWGN data augmentation since it provided 73% overall accuracy and acceptable performance on each target class. A very interesting pattern is the fact that all the models, also the ones without transformers, performed really well (F1 score $>$ 0.93) at recognizing "surprise" target class. This is clear evidence that the diagrams of surprised audio sample are highly recognizable. Another interesting aspect is the fact that MFCCs have proved to be definitely better that Mel Spectrograms since they outperformed them on the same test data both with or without data augmentation, as can be seen in figure 9.



(a) Mel Spectrograms, No Augmentation (b) Mel Spectrograms, Augmentation



(c) MFCCs, No Augmentation (d) MFCCs, Augmentation

Figure 9: Comparison of the confusion matrixes of all the models.

5 First pre-trained network: VGG16

5.1 Creation of the dataset

VGG16 is a convolution neural network developed by Karen Simonyan and Andrew Zisserman in 2014. It is a simple and widely used convnet architecture for ImageNet. It is composed of a serie of 3X3 convolutional layers and each pair or triple of convolutional layers is followed by a 2x2 maxpooling layer. Its architecture is represented in the figure 10. It was developed for the ILSVR(Imagenet)

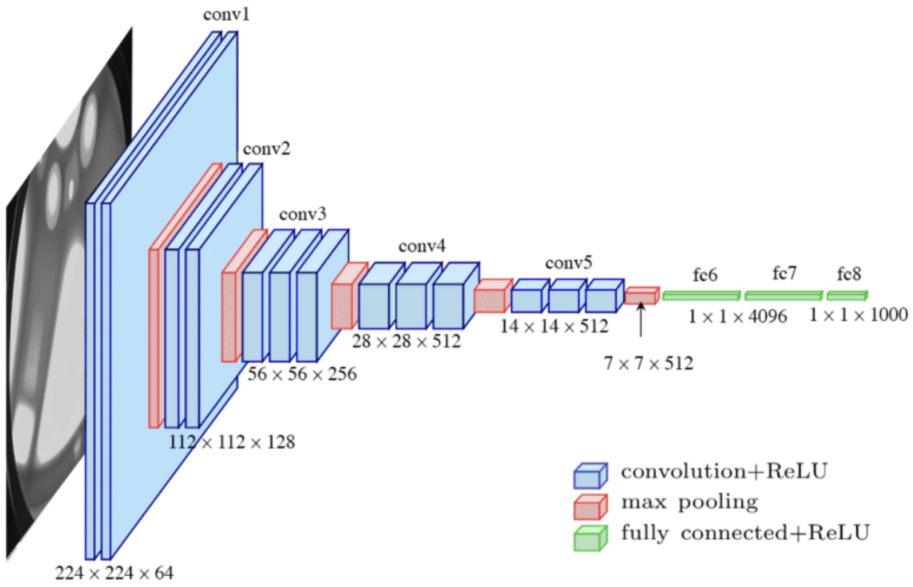


Figure 10: VGG16 architecture

competition in 2014 and it won it.
Since VGG16 is used for image classification, we need images. We divide the original dataset into 3 sets: train (80%), validation (10%) and test (10%). For each set and each audio, we compute the melspectrogram using the librosa library. The plot of each spectrogram is saved into a directory (the type of set and then the emotion). We do not plot the axis and the legend to keep only the values. The spectogram is ploted using a maximum and minimum values and an imposed color bar. Like this, the colors of each plot correspond to the same values.

5.2 Feature extraction

Feature extraction consists of using the representations learned by a previous network to extract interesting features from new samples. The convolutional base of the VGG16 is frozen and the features that it extracts are then sent as

input to a new classifier, which is trained from scratch.

5.2.1 Experiment 0

To have a starting point, we made a hyper-parameters tuning. This experiment is done by adding to the convolutional base two dense layers (whose activation function is relu) with a dropout layer between them. The output layer is a dense layer with 7 neurons (one for each class) and softmax as activation function). The number of neurons of the first dense layer is tested between 64 and 512 by step of 64. The second dense layer is tested between 32 and 256 by step of 32 and we tested 1e-3, 1e-4 and 1e-5 as learning rate values.
The optimal number of units in the first densely-connected layer is 512 and in the second it is 192. The optimal learning rate for the optimizer is 1e-05.
For each experiment, we use the Adam optimizer and the categorical crossentropy as loss function.

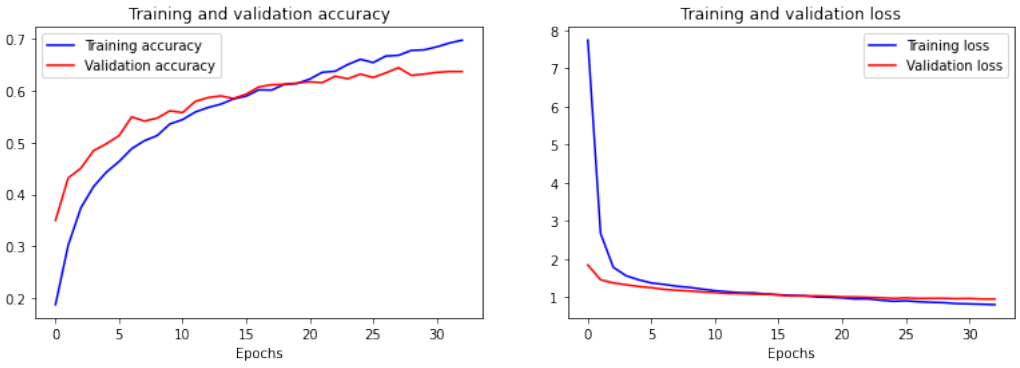
5.2.2 Experiment 1

The first experiment is done by using the optimal hyper-parameters found in the previous experiment. The architecture is given here:

Model: "vgg16_exp_1"

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	[(None, 224, 224, 3)]	0
vgg16 (Functional)	(None, 7, 7, 512)	14714688
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 512)	12845568
dropout (Dropout)	(None, 512)	0
dense (Dense)	(None, 192)	98496
dense (Dense)	(None, 7)	1351

Total params: 27,660,103
Trainable params: 12,945,415
Non−trainable params: 14,714,688



After 15 epochs, we observe overfitting: the accuracy on the training set is larger than the one on the validation set. And the accuracy on the validation

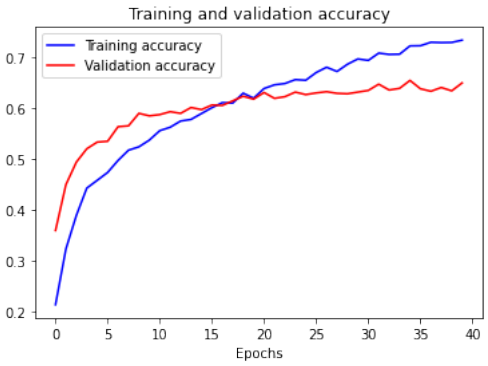
set does not increase significantly anymore, which means that the network stats to memorize the training data.
Since we use a early stop callback to stop the training when there is no improvement in the accuracy on the validation set with a patience of 5, we need to retrain the network with the best number of epochs. The values reported in the following table is the ones that we get with the network trained with the best number of epochs. The same procedure is applied for all the experiments of this section. We have to note that all the plots of the loss in this section do not have a very precise y-axis due to the first training loss (higher than other values). The precised values can be found in the corresponding jupyter notebook if needed.

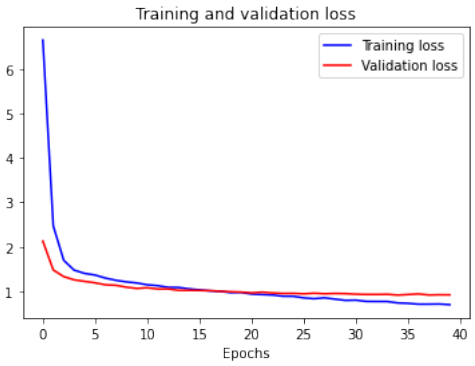
best epoch	validation accuracy	test accuracy	validation loss	test loss
15	0.5987	0.6146	1.0487	0.9991

	Precision	Recall	F1 Score	Support
Angry	0.65	0.82	0.72	187
Disgust	0.55	0.44	0.49	186
Fear	0.59	0.44	0.50	186
Happy	0.50	0.60	0.54	187
Neutral	0.58	0.62	0.60	177
Sad	0.66	0.59	0.63	187
Surprise	0.89	0.93	0.91	107
Accuracy			0.61	1217
Macro AVG	0.63	0.63	0.63	1217
Weighted AVG	0.61	0.61	0.61	1217

5.2.3 Experiment 2

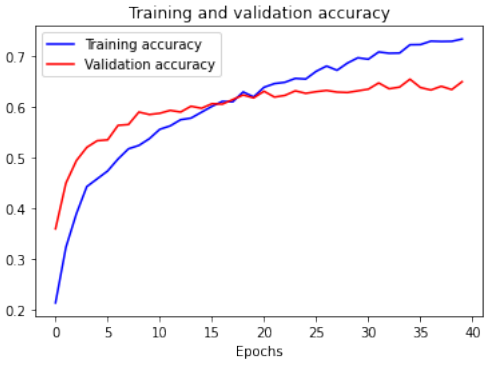
The second experiment is done by using the same classifier than the previous experiment and adding some data augmentation. The data augmentation consist in applying a horizontal random flip, random rotation and a random zoom on the images.

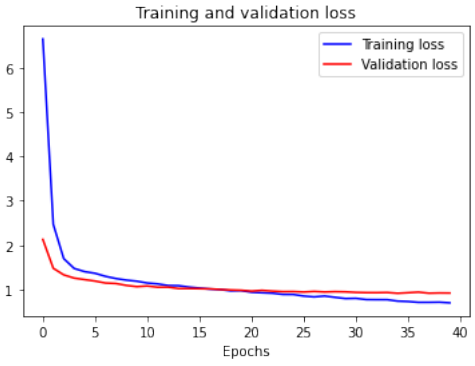




best epoch	validation accuracy	test accuracy	validation loss	test loss
15	0.5806	0.6145	1.0572	0.9853

	Precision	Recall	F1 Score	Support
Angry	0.59	0.85	0.69	187
Disgust	0.65	0.46	0.54	186
Fear	0.59	0.42	0.50	186
Happy	0.59	0.44	0.50	187
Neutral	0.54	0.72	0.62	177
Sad	0.59	0.60	0.59	187
Surprise	0.88	0.90	0.89	107
Accuracy			0.61	1217
Macro AVG	0.63	0.63	0.62	1217
Weighted AVG	0.62	0.61	0.60	1217





5.3 Fine tuning

Fine tuning consists in unfreezing a few of the top layers of a frozen model base used for feature extraction, and jointly training both the classifier we added and and these top layers. This is called "fine-tuning" because it slightly adjusts the more abstract representations of the model being reused, in order to make them more relevant for the problem at hand.

Fine tuning can be useful when the new dataset is large and very different from the original dataset. We can either train a CNN from scratch or fine-tuning a larger number of layers. Indeed, whe we have a lot of data, we do not run the risk of overfitting. And when the data is different, we could benefit from the weight initialization of a pre-trained network and fine-tune even the earlier layers.

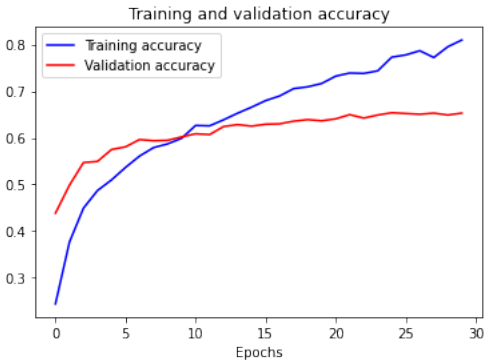
5.3.1 Experiment 3

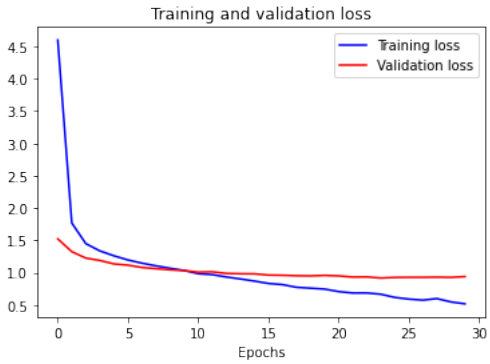
The third experiment is done by unfreezing the last convolutional block of vgg16. The architecture is given here:

Model: "vgg16_exp_3"

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 224, 224, 3)]	0
vgg16 (Functional)	(None, 7, 7, 512)	14714688
flatten_1 (Flatten)	(None, 25088)	0
dense_3 (Dense)	(None, 512)	12845568
dropout_1 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 192)	98496
dense_5 (Dense)	(None, 7)	1351

Total params: 27,660,103
Trainable params: 15,305,223
Non−trainable params: 12,354,880





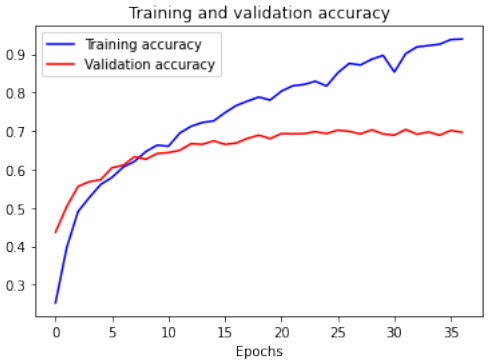
best epoch	validation accuracy	test accuracy	validation loss	test loss
9	0.6118	0.6549	1.0232	0.9444

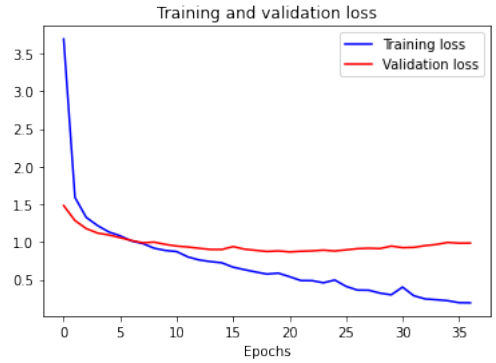
Table 1: VGG16 experiment 3

	Precision	Recall	F1 Score	Support
Angry	0.70	0.82	0.76	187
Disgust	0.61	0.59	0.60	186
Fear	0.65	0.44	0.52	186
Happy	0.62	0.57	0.60	187
Neutral	0.59	0.70	0.64	177
Sad	0.61	0.63	0.62	187
Surprise	0.87	0.97	0.92	107
Accuracy			0.65	1217
Macro AVG	0.66	0.67	0.66	1217
Weighted AVG	0.65	0.65	0.65	1217

5.3.2 Experiment 4

The fourth experiment is done by unfreezing the 2 last convolutional blocks of vgg16. The architecture is the same than previously, except that the total number of parameters is 27,660,103 (17,665,031 trainable and 9,995,072 non-trainable).



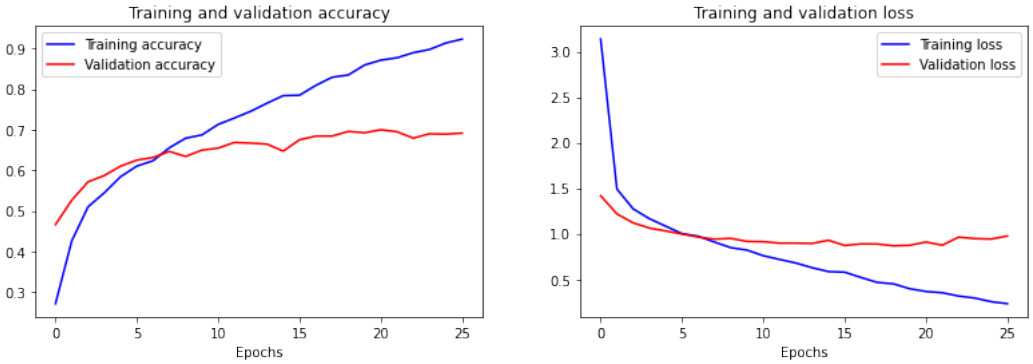


Best epoch validation accuracy	test accuracy	validation loss	test loss	
6	0.6102	0.6196	1.0355	0.9701

	Precision	Recall	F1 Score	Support
Angry	0.64	0.80	0.71	187
Disgust	0.57	0.52	0.54	186
Fear	0.48	0.50	0.49	186
Happy	0.55	0.56	0.55	187
Neutral	0.67	0.64	0.66	177
Sad	0.68	0.52	0.59	187
Surprise	0.88	0.93	0.90	107
Accuracy			0.62	1217
Macro AVG	0.64	0.64	0.63	1217
Weighted AVG	0.62	0.62	0.62	1217

5.3.3 Experiment 5

The fifth experiment is done by unfreezing the 3 last convolutional blocks of vgg16. The architecture is the same than previously, except that the total number of parameters is 27,660,103 (20,024,839 trainable and 7,635,264 non-trainable).

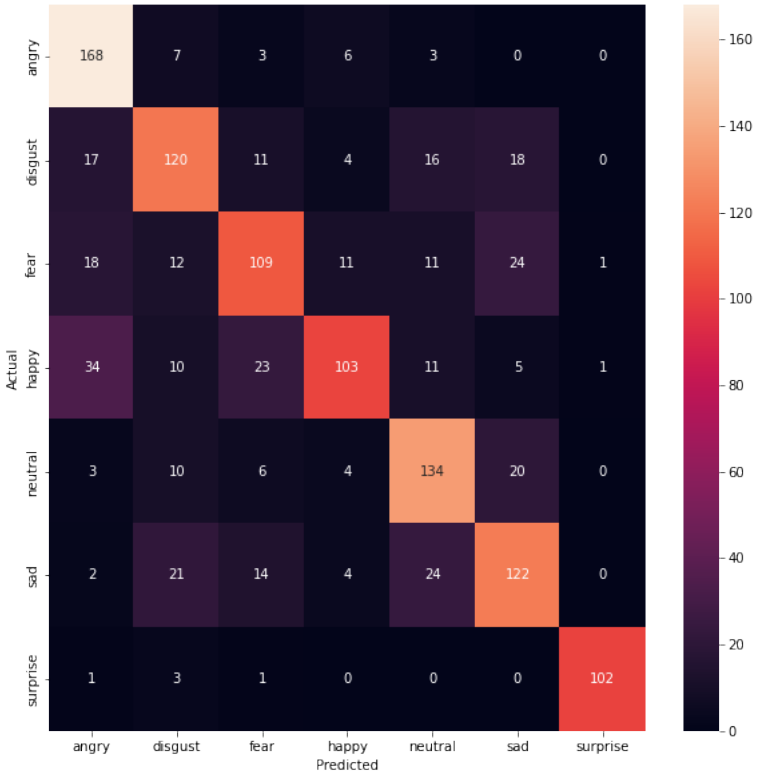


best epoch	validation accuracy	test accuracy	validation loss	test loss
6	0.6209	0.6417	0.9627	0.9215

	Precision	Recall	F1 Score	Support
Angry	0.67	0.84	0.75	187
Disgust	0.54	0.62	0.58	186
Fear	0.60	0.44	0.51	186
Happy	0.70	0.44	0.54	187
Neutral	0.61	0.66	0.63	177
Sad	0.58	0.67	0.62	187
Surprise	0.93	0.95	0.94	107
Accuracy			0.64	1217
Macro AVG	0.66	0.66	0.65	1217
Weighted AVG	0.65	0.64	0.63	1217

5.4 Discussion of the results obtained with VGG16

First of all, we observed that the best accuracy we obtained for the test accuracy is 0.6549 (experiment 3: fine tuning with the last convolutional block of VGG16 unfreezed). We also reached an accuracy on the test set of 0.7050 with the network from the experiment 4 (fine tuning with the last convolutional block of VGG16 unfreezed), which was the best accuracy we reached during some other random the experiments. The corresponding confusion matrix is given below. This difference between the accuracy reported in the table of the experiment 4 Table **??** and 0.7050 can be explained by the fact that the weights are initialized randomly and this can affect the final results.



Another observation it that, by adding some data augmentation, we do not observe significant improvement. Indeed, data augmentation is technique used to artificially increase the size of a dataset by generating new, modified versions of existing data. And in our case, we already have a large number of data. More over, our images are spectograms. There are all constructed in the same way. So the features the network finds may not be flipped or rotated between spectograms.
We observed that the fine tuning improved the results. The ImageNet dataset, on which VGG16 was trained, is a large-scale image classification dataset that contains over 14 million images and 1,000 different object classes. These classes are for example animals, objects, natural phenomena, food... And our dataset is very different. So we need to retained the last convolutional layers of VGG16.

6 Fine Tuning of a Transformer pre-trained network: Wav2Vec2

6.1 Introduction

The Wav2Vec2 model was proposed by researchers at Facebook AI in [7] and it is an extension of the original wav2vec approach. The original model is based on a transformer neural network architecture to learn representations of speech audio that can be used for a variety of tasks, such as speech recognition, speaker identification, and language modeling. The basic idea behind wav2vec is to train a transformer-based model to predict the next audio sample in a sequence, given the previous samples. Once the model is trained, it can be used to encode an input speech signal into a fixed-dimensional vector representation, called an embedding.

Wav2vec 2.0 has been shown to achieve state-of-the-art performance on a variety of speech recognition benchmarks and is considered as one of the most powerful unsupervised models for speech recognition today.

The model first processes the raw waveform of the speech audio with a multilayer convolutional neural network to get the audio representations. The **CNN encoder** consists of several blocks containing a temporal convolution followed by layer normalization and a GELU activation function. The GELU activation function is a non-linear function that is used to introduce non-linearity into the neural network, allowing it to learn more complex and powerful representations of the input data. The GELU function is defined as:

$$GELU(x) = x * \varphi(x)$$

where x is the input to the activation function, and $\varphi(x)$ is the probability density function of a standard normal distribution. The use of the GELU activation function has been shown to improve the performance of neural network models, going to model the distribution of the data more effectively.

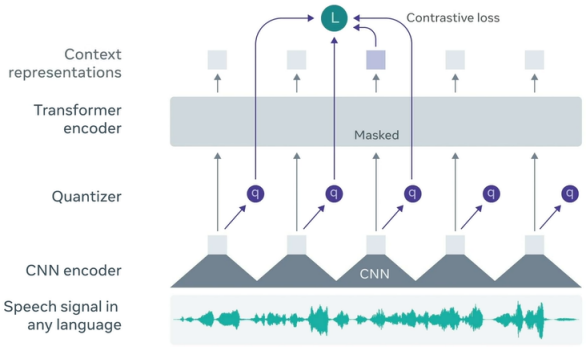


Figure 11: Wav2Vec2 model

The **quantizer** is used to convert the continuous representations learned from the model into discrete representations. This is done by dividing the continuous space into a fixed number of discrete "bins," and then mapping each continuous vector to the nearest bin. The main motivation for using a quantizer is to make the learned representations more compact and efficient to store and use. The quantizer used in Wav2vec 2.0 is an adaptive quantizer, which means that the size and location of the bins can be adjusted during training to optimize the quality of the learned representations.

The **transformer** adds information from the entire audio sequence. Finally, the output of the transformer is used to solve a contrastive task, it's a type of unsupervised learning problem where the goal is to learn representations of data by comparing and contrasting different examples. This task requires the model to identify the correct quantized speech units for the masked positions.

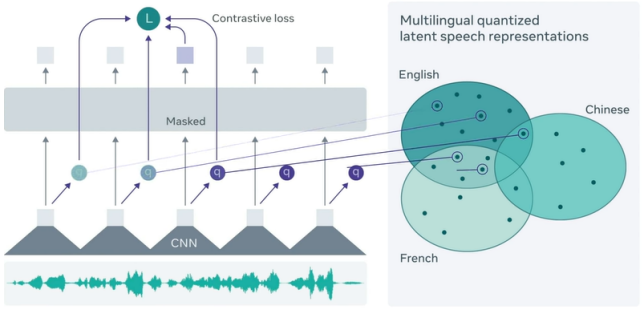


Figure 12: Example of contrastive loss on languages classification

6.2 Fine Tuning

We have fine-tuned the Wav2Vec 2.0 (base) model, built on the Hugging Face Transformers library, this involves training the model on a new dataset, while using the pre-trained weights as the initial weights for the model, so it should be able to quickly learn the new task with less data and fewer computational resources.

To perform the audio classification task with this model we need to add a Classification-Head on top to output a probability distribution of all classes for each input audio sample, as a Keras layer. This implementation is based on this example provided by Keras documentation.

In order to use this model for emotion classification, we need to pre-process the audio sample. This is done by download from Hugging Face the Feature Extractor that corresponds to the model architecture that we use. The feature extractor is in charge of preparing input features for the model, the output of the feature extractor is an encoded vector sequence. Note that we need to resample the input audio to 16kHz, since the base model is pretrained on 16kHz sample speech audio and all the input are truncated at 3 second.

```
Model: "Wac2Vec2_AudioClassification"
_________________________________________________________________________________________________
 Layer (type)                   Output Shape              Param #   Connected to
==================================================================================================
 attention_mask (InputLayer)    [(None, 48000)]           0         []

 input_values (InputLayer)      [(None, 48000)]           0         []

 tf_wav2_vec2_for_audio_classif   (32, 7)                 94377095  ['attention_mask[0][0]',
 ication (TFWav2Vec2ForAudioCla                                      'input_values[0][0]']
 ssification)

==================================================================================================
Total params: 94,377,095
Trainable params: 94,377,095
Non-trainable params: 0
_________________________________________________________________________________________________
```

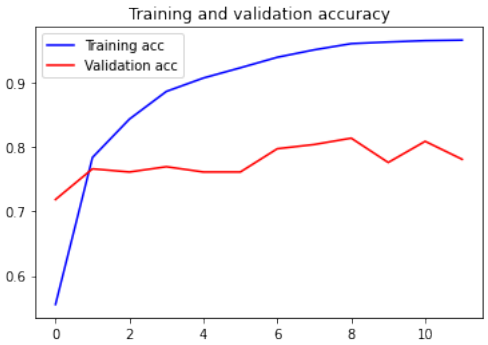
6.3 Evaluation

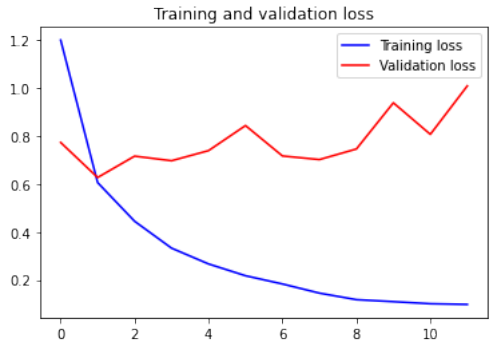
In this section will be analyzed the results obtained fine tuning the transformer Model.

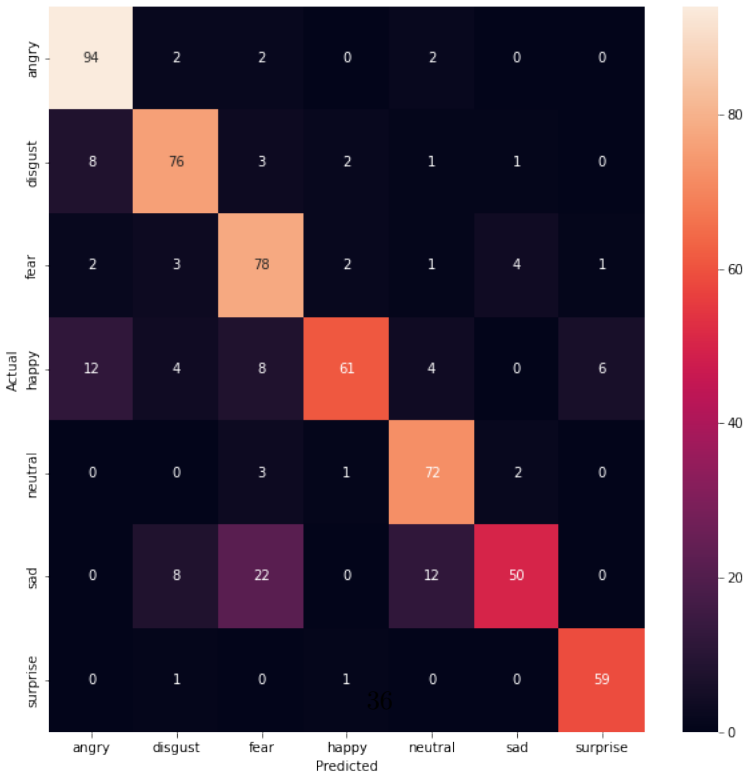
The dataset is splitted in 90% train 5% validation and 5% test. The model has been optimized using Adam optimizer with 1e-5 learning rate, and the *SparseCategoricalCrossentropy* is used as loss function. It has been trained with a maximum number of epochs of 30 and with early stopping monitoring the validation loss. Since the limitation on the GPU usage and the memory available, we need to use gradient checkpointing to save memory at the expense of slower backward pass.

We can see that this model overfit after a few epoch, so we used Early Stopping to monitor the performance during the training and stop if the validation loss is not improving with a patience of 10. It's important to keep in mind that some overfitting is natural and expected, especially with large and powerful models like transformers. We can see how this model, with a few epochs of training can perform better than all other tests performed previously.

	Precision	Recall	F1 Score	Support
Angry	0.81	0.94	0.87	100
Disgust	0.81	0.84	0.82	91
Fear	0.67	0.86	0.75	91
Happy	0.91	0.64	0.75	95
Neutral	0.78	0.92	0.85	78
Sad	0.88	0.54	0.67	92
Surprise	0.89	0.97	0.93	61
Accuracy			0.81	608
Macro AVG	0.82	0.82	0.81	608
Weighted AVG	0.82	0.81	0.80	608







7 Conclusion

The single CNN and the 2 parallel CNNs have similar accuracy. So the addition of a CNNs block in parallel is not very effective in improving the performance. We observed a slight improvement when using the Transformer Encoder Block but at the cost of having a definetely more complex network. The data augmentation performed to augment the audio sample with Gaussian white noise proved to be very effective since it improved the generalization capability of each of the model on which we applied it.

Using VGG16 does not give better results than the last model trained from scratch (2 CNNs and encoder) even if it is a pre-trained network. Indeed, the data set is very different than the one used to train VGG16. Using fine-tuning instead of feature extraction improved the performance but is not enough to perform better the last model trained from scratch. Moreover, the data augmentation here does not boost the results. The data augmentation is different, it is performed on the images and not directly on the original data.

The model obtained fine-tuning Wav2Vec2 has achieved the best accuracy on the training set. We can see how in a very few epochs the network outperforms the other models, but with some more epochs overfit, learning too well on the training data and as a result, it performs poorly on unseen data. This happens because the model is too complex and is able to learn the noise in the training data.

References

[1] Wootaek Lim, Daeyoung Jang, and Taejin Lee. Speech emotion recognition using convolutional and recurrent neural networks. In *2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, pages 1–4, 2016.

[2] Sadia Sultana, M. Zafar Iqbal, M. Reza Selim, Md. Mijanur Rashid, and M. Shahidur Rahman. Bangla speech emotion recognition and cross-lingual study using deep cnn and blstm networks. *IEEE Access*, 10:564–578, 2022.

[3] Sandeep Kumar Pandey, H. S. Shekhawat, and S. R. M. Prasanna. Deep learning techniques for speech emotion recognition: A review. In *2019 29th International Conference Radioelektronika (RADIOELEKTRONIKA)*, pages 1–6, 2019.

[4] Neri E. Cibau, Enrique Marcelo Albornoz, and Hugo Leonardo Rufiner. Speech emotion recognition using a deep autoencoder. 2013.

[5] John Lorenzo Bautista, Yun Kyung Lee, and Hyun Soon Shin. Speech emotion recognition based on parallel cnn-attention networks with multi-fold data augmentation. *Electronics*, 11(23), 2022.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[7] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations, 2020.