AEI: Artificial 'Emotional' Intelligence

Namita Ramesh
Dec 13, 2019 · 15 min read ★

> It all began about 2,000 years ago when Plato wrote, "All learning has an emotional base."

One might think that over 200,000 years of evolution would make humans masters of emotions. Yet, we live in a world where people, irrespective of age or maturity, often make errors in emotional judgment. Clarity in identifying emotions is key to social behaviors such as smooth communication and building long-lasting relationships.

What makes identifying emotions challenging for humans?

We often struggle to express our emotions and articulate our feelings. emotions come in many different degrees, qualities, and intensities. In addition, our experiences are often comprised of multiple emotions at once, which adds another dimension of complexity to our emotional experience.

The icing on the cake is, however, Emotional Bias. With a spectrum as variant as the range of emotions, there is bound to be bias. This is where the problem gets interesting for us as data scientists- we love a good 'Bias-Variance' problem!

Enter, your friendly, unbiased neighborhood Emotion Detector Bot. Gone are the days when the only thing separating man and machine was emotional intelligence. Emotion Recognition or Artificial 'Emotional' Intelligence is now a $20 billion field of research with applications in many different industries.

Across industries, artificial emotional intelligence can work in a number of ways. For example, AI can monitor a user's emotions and analyze them to achieve a certain outcome. This application would prove extremely useful in enhancing automated

Customer Service calls. AI can also use emotional readings as part of decision making, for example in marketing campaigns. Advertisements can be changed based on consumer reactions.

. . .

RAVDESS Data Set

The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) contains 7356 files (total size: 24.8 GB). The database contains 24 professional actors (12 female, 12 male), vocalizing two lexically-matched statements in a neutral North American accent. Speech includes calm, happy, sad, angry, fearful, surprise, and disgust expressions. Each expression is produced at two levels of emotional intensity (normal, strong), with an additional neutral expression. All conditions are available in three modality formats: Audio-only (16bit, 48kHz .wav), Audio-Video (720p H.264, AAC 48kHz, .mp4), and Video-only (no sound).

Each of the 7356 RAVDESS files has a unique filename. The filename consists of a 7-part numerical identifier (e.g., 02–01–06–01–02–01–12.mp4). These identifiers define the stimulus characteristics:

Filename identifiers:

- Modality (01 = full-AV, 02 = video-only, 03 = audio-only)
- Vocal channel (01 = speech, 02 = song).
- Emotion (01 = neutral, 02 = calm, 03 = happy, 04 = sad, 05 = angry, 06 = fearful, 07 = disgust, 08 = surprised).
- Emotional intensity (01 = normal, 02 = strong). NOTE: There is no strong intensity for the 'neutral' emotion.
- Statement (01 = "Kids are talking by the door", 02 = "Dogs are sitting by the door").
- Repetition (01 = 1st repetition, 02 = 2nd repetition).
- Actor (01 to 24. Odd numbered actors are male, even numbered actors are female).

. . .

Approach

Given the diversity of our data set in terms of data types — speech, songs and video — we decided to separate audio and audio-video files and model them separately to identify emotions.

You will find the individual details of our two distinct models and the overarching conceptual highlight in the coming sections.

Emotion Detection from Videos

Have you ever thought someone was angry at you, but it turned out you were just misreading their facial expression? There are 7 basic emotions that our faces can emote. Now, imagine what combinations of these would be necessary to emote ‘muted happiness’. We cannot put a number on the different kinds of emotions one can express given different permutations and combinations of our seven basic emotions.

Source — https://egyptinnovate.com/en/success-stories/affectiva-company-humanizing-technology

The ability to read emotions from faces is a very important skill. One might even call it a superpower. It is this skill that has enabled and facilitated human interactions since time immemorial.

Subconsciously we see, label, make predictions, and recognize patterns all day every day. But how do we do that? How is it that we can interpret everything that we see?

It took nature over 500 million years to create a system to do this. The collaboration between the eyes and the brain, called the primary visual pathway, is the reason we can make sense of the world around us.

The **deeply complex hierarchical structure** of neurons and connections in the brain play a major role in this process of remembering and labeling objects. In the beginning, we were taught the name of the objects around us. We learned by examples that were given to us. Slowly but surely, we started to recognize things more and more often in our environment. They became so common that the next time we saw them, we would instantly know what the name of this object was. They became part of our '**model**' of the world.

But how do modern machines recognize emotions from facial expressions?

Convolutional Neural Networks

Similar to how a child learns to recognise objects, we need to train an algorithm on millions of pictures before it is able to perceive the input and make predictions for unseen images.

Computers 'see' in a different way than we do. Their world consists of only numbers. Every image can be represented as 2-dimensional arrays of numbers, known as pixels.

Convolutional Neural Network (CNN) is a specific type of Artificial Neural Network that teaches an algorithm how to recognize objects/features in images.

Here's how we leveraged the power of CNN in our project.

Defining the CNN model

Keras was used to create a Sequential Convolutional Network — neural network with a linear stack of layers. This network has the following components:

Convolutional Layers: These layers are the building blocks of our network. These compute dot product between input image X and a set of Kj learnable filters. Each filter Kj sized k1 × k2 moves across the input space performing the convolution with local subblocks of inputs, providing Yj, the feature maps ($Yj=\sum X \times Kj+Bj$, where B is the bias term).

Activation functions: We use activation functions to make our output non-linear. In the case of a Convolutional Neural Network, the output of the convolution will be passed through the activation function. In this project, we will resort to the use of two functions — Relu and Sigmoid.

Pooling Layers: These layers will downsample the operation along the dimensions. This helps reduce the spatial data and minimize the processing power that is required.

Dense layers: These layers are present at the end of a C.N.N. They take in all the feature data generated by the convolution layers and do the decision making.

Dropout Layers: randomly turns off a few neurons in the network to prevent over fitting.

Batch Normalization: normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. This speeds up the training process.

The network takes in an input of a two-dimensional array of size 256*256 and predicts one of the eight emotions present in the data set.

Here is a summary of the CNN model :

```
batch_normalization_5 (Batch (None, 121, 121, 128)     512
_________________________________________________________________
flatten_1 (Flatten)          (None, 1874048)           0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                59969568
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
```

```
_________________________________________________________________
___
dense_3 (Dense)              (None, 8)                 264
=================================================================
===
Total params: 60,248,824
Trainable params: 60,248,344
Non-trainable params: 480
_________________________________________________________________
___
```

Video_Emotion_Train.ipynb hosted with ❤️ by GitHub view raw

Parsing Video to obtain images

The first step would be to parse the video file into a set of image frames that we use to train the model. We use the cv2 library to capture images from a video. The VideoCapture function reads the video file and converts it into a sequence of image frames.

Each frame obtained will contain a two-dimensional array of integers containing information of the image. The images are composed of pixels and these pixels are channels of multiple arrays of numbers. Colored images have three color channels — red, green, and blue — and each channel is represented by a grid. Each cell in the grid stores a number between 0 and 255 which denotes the intensity of that cell. To capture a different expression each time, we pass every 20th frame into our training model.

What you see (L) vs what computer sees (R)

After extracting the image data, we resize the images to 256*256 to retain as much information as possible to enhance the accuracy of the model. We converted these

images to gray-scale, so that there is only one channel thereby reducing complexity.

```
        frame=cv2.resize(frame, (256,256))
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        frame = np.reshape(frame, [frame.shape[0], frame.shape[1], 1])

        x.append(frame)
        y.append(file_emotion)

```

We then append the image data to obtain a generator object which we shall explain further about in the following paragraphs.

Obtain labels to identify Emotions

We define emotions in a dictionary as shown in the code below.

```
#Assign the Emotion Dictionary
emotion_dict={'neutral':1, 'calm':2, 'happy':3, 'sad':4 , 'angry':5, 'fearful':6, 'disgust':7, 'surprised':8}
```

Emotion Dictionary

> *Sample Video File : '01–01–03–02–02–02–01.mp4'*

All the video files have the emotion listed in the filename as shown above. We split the filename and use the predefined emotion dictionary to obtain the labels for each video file.

Training the CNN Model

For small, simplistic datasets it's perfectly acceptable to use Keras' .fit function. However, large datasets such as ours are often too large to fit in memory. Data augmentation was performed to avoid overfitting and increase the model's ability to generalize. In those situations we need to utilize Keras' .fit_generator function. The .fit_generator function accepts batches of data, performs backpropagation, and then updates the weights in our model. This process is repeated until we have reached the desired number of epochs. You'll notice we now need to supply a steps_per_epoch parameter when calling .fit_generator (the .fit method had no such parameter).

Keras data generator is meant to loop infinitely. Thus, Keras cannot determine when one epoch starts and a new epoch begins. Therefore, we compute the steps_per_epoch value as the total number of training data points divided by the batch size. Once Keras hits this step count it knows that it's a new epoch.

```
model.fit_generator(video_generator(test_on), steps_per_epoch=25,epochs = 50,verbose = 1)
model.save('Video_Emotion_Detection_50.hdf5')
```

```
./Actor_12/01-01-06-02-01-02-12.mp4
 5/25 [=====>........................] - ETA: 2:55 - loss: 6.9040 - acc: 0.570001-01-06-02-01-01-12.mp4
[0, 0, 0, 0, 0, 1, 0, 0]
./Actor_12/01-01-06-02-01-01-12.mp4
 6/25 [======>.......................] - ETA: 2:44 - loss: 7.2230 - acc: 0.550001-01-06-02-02-01-12.mp4
[0, 0, 0, 0, 0, 1, 0, 0]
./Actor_12/01-01-06-02-02-01-12.mp4
 7/25 [=======>......................] - ETA: 2:34 - loss: 7.3938 - acc: 0.539301-01-06-02-02-02-12.mp4
[0, 0, 0, 0, 0, 1, 0, 0]
./Actor_12/01-01-06-02-02-02-12.mp4
 8/25 [========>.....................] - ETA: 2:25 - loss: 7.4223 - acc: 0.537501-01-07-01-01-01-12.mp4
[0, 0, 0, 0, 0, 0, 1, 0]
./Actor_12/01-01-07-01-01-01-12.mp4
 9/25 [=========>....................] - ETA: 2:16 - loss: 7.5773 - acc: 0.527801-01-07-01-02-01-12.mp4
[0, 0, 0, 0, 0, 0, 1, 0]
./Actor_12/01-01-07-01-02-01-12.mp4
10/25 [==========>...................] - ETA: 2:07 - loss: 7.6216 - acc: 0.525001-01-07-01-01-02-12.mp4
[0, 0, 0, 0, 0, 0, 1, 0]
./Actor_12/01-01-07-01-01-02-12.mp4
11/25 [===========>..................] - ETA: 1:59 - loss: 7.5853 - acc: 0.527301-01-07-02-01-02-12.mp4
[0, 0, 0, 0, 0, 0, 1, 0]
./Actor_12/01-01-07-02-01-02-12.mp4
12/25 [============>.................] - ETA: 1:50 - loss: 7.3547 - acc: 0.541701-01-07-01-02-02-12.mp4
[0, 0, 0, 0, 0, 0, 1, 0]
./Actor_12/01-01-07-01-02-02-12.mp4
13/25 [=============>................] - ETA: 1:40 - loss: 7.1747 - acc: 0.552901-01-07-02-01-01-12.mp4
[0, 0, 0, 0, 0, 0, 1, 0]
./Actor_12/01-01-07-02-01-01-12.mp4
14/25 [==============>...............] - ETA: 1:32 - loss: 6.9776 - acc: 0.565201-01-08-01-01-01-12.mp4
[0, 0, 0, 0, 0, 0, 0, 1]
./Actor_12/01-01-08-01-01-01-12.mp4
15/25 [================>.............] - ETA: 1:23 - loss: 6.8605 - acc: 0.572501-01-07-02-02-02-12.mp4
[0, 0, 0, 0, 0, 0, 1, 0]
./Actor_12/01-01-07-02-02-02-12.mp4
16/25 [=================>............] - ETA: 1:15 - loss: 6.6824 - acc: 0.583601-01-08-01-01-02-12.mp4
[0, 0, 0, 0, 0, 0, 0, 1]
./Actor_12/01-01-08-01-01-02-12.mp4
17/25 [===================>..........] - ETA: 1:06 - loss: 6.5725 - acc: 0.590401-01-07-02-02-01-12.mp4
[0, 0, 0, 0, 0, 0, 1, 0]
./Actor_12/01-01-07-02-02-01-12.mp4
18/25 [====================>.........] - ETA: 58s - loss: 6.4528 - acc: 0.5979 01-01-08-01-02-01-12.mp4
[0, 0, 0, 0, 0, 0, 0, 1]
./Actor_12/01-01-08-01-02-01-12.mp4
19/25 [=====================>........] - ETA: 50s - loss: 6.4297 - acc: 0.599301-01-08-01-02-02-12.mp4
[0, 0, 0, 0, 0, 0, 0, 1]
./Actor_12/01-01-08-01-02-02-12.mp4
20/25 [======================>......] - ETA: 41s - loss: 6.3088 - acc: 0.606901-01-08-02-01-01-12.mp4
[0, 0, 0, 0, 0, 0, 0, 1]
./Actor_12/01-01-08-02-01-01-12.mp4
21/25 [=======================>.....] - ETA: 33s - loss: 6.2184 - acc: 0.612501-01-08-02-02-01-12.mp4
[0, 0, 0, 0, 0, 0, 0, 1]
./Actor_12/01-01-08-02-02-01-12.mp4
22/25 [========================>....] - ETA: 25s - loss: 6.1362 - acc: 0.617601-01-08-02-02-02-12.mp4
[0, 0, 0, 0, 0, 0, 0, 1]
./Actor_12/01-01-08-02-02-02-12.mp4
23/25 [=========================>...] - ETA: 16s - loss: 6.0149 - acc: 0.625201-01-08-02-01-02-12.mp4
[0, 0, 0, 0, 0, 0, 0, 1]
./Actor_12/01-01-08-02-01-02-12.mp4
24/25 [==========================>..] - ETA: 8s - loss: 5.9314 - acc: 0.6304 02-01-01-01-01-01-12.mp4
[1, 0, 0, 0, 0, 0, 0, 0]
./Actor_12/02-01-01-01-01-01-12.mp4
25/25 [==============================] - 210s 8s/step - loss: 5.9181 - acc: 0.6312
```

After trying out different iterations and switching up the model parameters, we zeroed in on 50 epochs and 25 steps per epoch. We obtained an accuracy of 0.55 to 0.65 for this model.

Testing the CNN Model

To test the CNN model, we use the Keras function test_on_batch(). Similar to how we trained the model, we capture every 20th frame in the video file, convert the image data into Grayscale and reshape the images into 256*256 arrays. The model returns an array of eight numbers corresponding to an emotion. We obtain the predicted emotion by determining the highest number in this array.

We tested the model on two actors and obtained an accuracy of 0.6 on these results.

Model Prediction

After testing the model, we wanted to test the model on an unseen video file. We passed a video with a 'happy' emotion tag through the model.

What we receive as the output of the model is a weighted array of emotions. We run the code given below to pick the best possible outcome for this model.

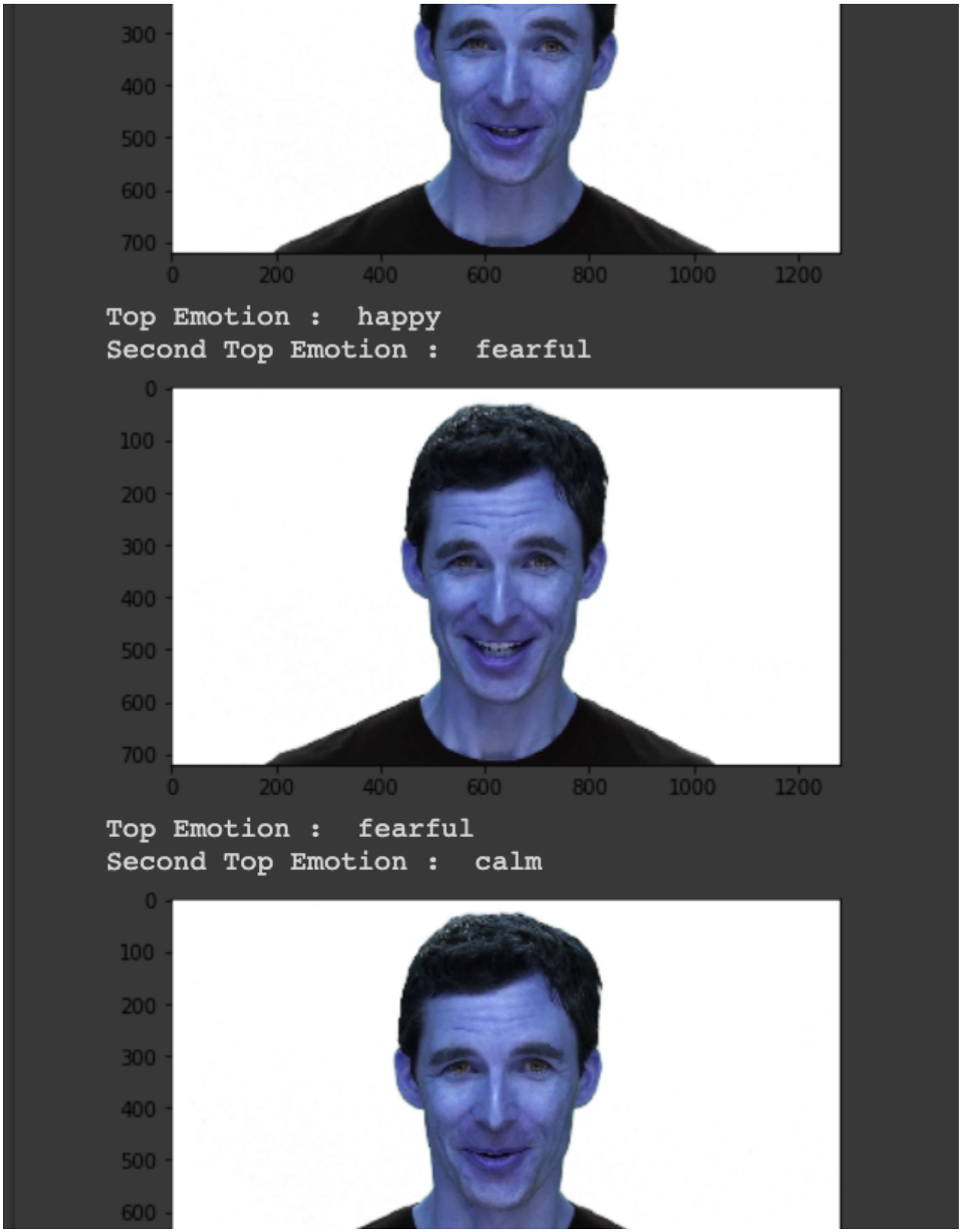
```
#Obtain Prediction from array of pixels
pred_array=model.predict(x)

#Find Top Two Predicted Emotions
predicted_class=np.argpartition(pred_array[0], -4)[-4:][0] + 1
predicted_class2=np.argpartition(pred_array[0], -4)[-4:][1] + 1

label_map = dict((v,k) for k,v in emotion_dict.items())
predicted_label = label_map[predicted_class]
print("Top Emotion : ",predicted_label)
predicted_label2 = label_map[predicted_class2]
print("Second Top Emotion : ",predicted_label2)

```

Here is an example of the model predicting the top two emotions for every 20th frame in the video file.



The outcome of each frame is combined to give the final predicted outcome for the video clip.

Transfer Learning Approach

In addition to using CNN, we experimented with Transfer learning to determine if we could obtain a higher accuracy by training a pre-trained model on our data set. Transfer

Learning is a machine learning method where a model developed for a certain predictive problem is re-purposed as the starting point for a model on a second task.

This model is trained on FER data which predicts seven emotions.

```
1  model.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_54 (Conv2D)           (None, 46, 46, 32)        320
_________________________________________________________________
conv2d_55 (Conv2D)           (None, 44, 44, 64)        18496
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 22, 22, 64)        0
_________________________________________________________________
conv2d_56 (Conv2D)           (None, 20, 20, 128)       73856
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 10, 10, 128)       0
_________________________________________________________________
conv2d_57 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_58 (Conv2D)           (None, 4, 4, 7)           903
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 1, 1, 7)           791
_________________________________________________________________
flatten_10 (Flatten)         (None, 7)                 0
_________________________________________________________________
activation_10 (Activation)   (None, 7)                 0
=================================================================
Total params: 241,950
Trainable params: 241,950
Non-trainable params: 0
_________________________________________________________________
```

We removed the last five layers and added two Convolutional layers and an Activation layer as shown below in the summary.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model_1 (Model)              (None, 4, 4, 128)         240256
=================================================================
Total params: 240,256
Trainable params: 240,256
Non-trainable params: 0
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model_1 (Model)              (None, 4, 4, 128)         240256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 4, 4, 8)           1032
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 1, 1, 8)           1032
_________________________________________________________________
flatten_1 (Flatten)          (None, 8)                 0
_________________________________________________________________
activation_1 (Activation)    (None, 8)                 0
=================================================================
Total params: 242,320
Trainable params: 242,320
Non-trainable params: 0
_________________________________________________________________
```

We trained this model on the video dataset and obtained an accuracy of 0.35. We plan to work on increasing the accuracy of this model and determine if this can beat the accuracy of our initial CNN model.

. . .

Identifying Emotions from Audio Signals

The way humans process sound is incredibly complicated and there are a lot of factors that go into how an emotion is perceived from an audio clip. The gender of a person, the inflections in their tone, even the type of words being used affect the way we perceive what is being said.

The audio files in our dataset include 3 second audio clips, both speech and songs. For the scope of this project, we restricted ourselves to just the speech clips. The audio was mostly free from any sort of background noise and was recorded in a controlled environment.

We encountered many challenges when it came to making a model that could understand emotions from the audio clips available to us. The first and the biggest one for us was to figure out what features we would need in order to make our model. This is was a very domain-specific task and we needed to understand sound and its underlying properties and figure out what features can help identify emotions properly.

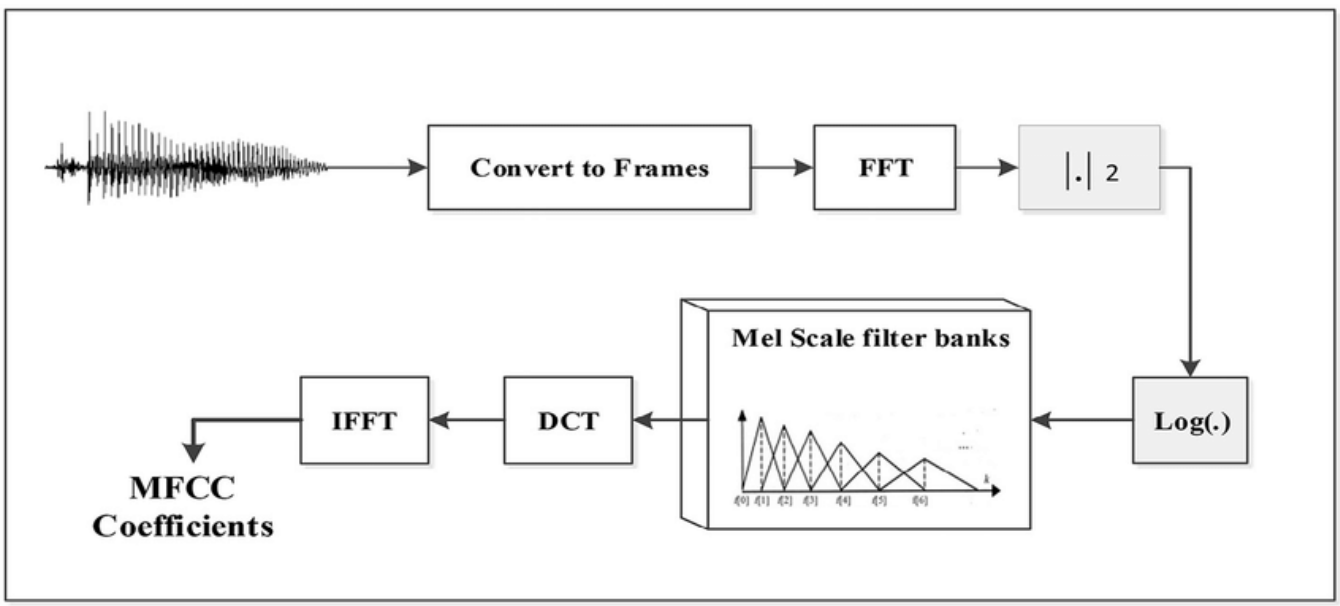
MFCC:

The Mel Frequency Cepstrum (MFC) is a short-term power-spectrum of data and is especially useful for speech analysis. Sounds emitted by humans are influenced by the shape of the vocal tract (including the vocal cords, larynx, tongue, teeth, etc.). In the most basic sense, the Mel Frequency Cepstrum numerically represents this vocal passage. The Mel-scale aims to mimic the non-linear human ear perception of sound, by being more discriminative at lower frequencies and less discriminative at higher frequencies. It scales the frequency so that it matches closely with what the human ear can hear (humans are better at identifying small changes in speech at lower frequencies).

Mel Frequency Cepstral Coefficients (MFCCs) are a set of coefficients that collectively make up the Mel Frequency Cepstrum. As a high-level overview, the following steps are involved in the calculation of the MFCCs *(taken from Wikipedia)*:

- Take the Fourier transform of (a windowed excerpt of) a signal.
- Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows.
- Take the logs of the powers at each of the mel frequencies.
- Take the discrete cosine transform of the list of mel log powers, as if it were a signal.
- The MFCCs are the amplitudes of the resulting spectrum.

Delta MFCCs and delta-delta MFCCs — which are the first and second-order derivatives of the MFCCs — are also significant features for our predictive model.



Source — https://www.researchgate.net/figure/Extraction-Mel-frequency-cepstral-coefficients-MFCC-from-the-audio-recording-signals_fig1_289375827

Data Preprocessing:

For all our audio processing we used a python package called librosa which is really helpful for music and sound analysis.

```
for file in actor_folder:
    file_name = os.path.join(path, file['title'])
    y, sr = librosa.load(file_name)

    #generating the mfcc, delta mfcc and delta delta mfcc
    mfcc = librosa.feature.mfcc(y=y, sr=sr, hop_length=512, n_mfcc=13)
    delta_mfcc = librosa.feature.delta(mfcc)
    delta_delta_mfcc = librosa.feature.delta(mfcc, order = 2)

    #aggregating the features by calculating the min, max, standard deviation and mean
    min_mfcc = np.min(mfcc, axis = 1)
    min_delta_mfcc = np.min(delta_mfcc, axis = 1)
    min_delta_delta_mfcc = np.min(delta_delta_mfcc, axis = 1)

    max_mfcc = np.max(mfcc, axis = 1)
    max_delta_mfcc = np.max(delta_mfcc, axis = 1)
    max_delta_delta_mfcc = np.max(delta_delta_mfcc, axis = 1)

    sd_mfcc = np.std(mfcc, axis = 1)
    sd_delta_mfcc = np.std(delta_mfcc, axis = 1)
    sd_delta_delta_mfcc = np.std(delta_delta_mfcc, axis = 1)

    mfcc = mfcc.T
    mfcc = np.mean(mfcc, axis = 0)
    delta_mfcc = delta_mfcc.T
    delta_mfcc = np.mean(delta_mfcc, axis = 0)
    delta_delta_mfcc = delta_delta_mfcc.T
    delta_delta_mfcc = np.mean(delta_delta_mfcc, axis = 0)
```

APM_Audio.py hosted with ❤️ by GitHub view raw

We loaded the audio files using **librosa** with a sampling rate of 22050 Hz. Each file was divided into 157 frames. A frame is short slice of a time series used for analysis purposes. Librosa has a function for generating MFCCs from an audio file. For each frame, we only computed the first 13 MFCCs since they capture most of the information required for our analysis. Even though higher order MFCCs do contain further spectral

details of our audio files, they add extra complexity to the model which is often undesired.

The function returns the 13 MFCCs for each of the 157 frames in the audio. These were aggregated for each of the frames. The mean, maximum, minimum and standard deviations over the frames for each MFCCs were used. The same aggregations were done for the delta and the delta delta coefficients.

Another feature we extracted from the audio files was the root mean squared energy of the audio.

To each of the audio file, the emotion label, emotion intensity, gender and actor number were extracted from the file names.

Modelling Approach:

The first thing we did before modeling was to divide the datasets into train and test sets.

Normally this is done randomly. In our case, we decided to manually split the data by taking the first 20 actors in the training set and the last 4 actors in the test set. This is because randomly splitting the actors would cause a data leakage problem. Manually isolating certain actors helped avoid this.

The size of our data restricted us in terms of training a neural network from scratch. Ideally, artificial neural networks require millions of data points to train an accurate algorithm. However, we had to work with a few thousands of audio signals to train our models.

The following is a summary of the model we built:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 157, 128)          768
_________________________________________________________________
activation_1 (Activation)    (None, 157, 128)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 157, 128)          0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 19, 128)           0
```

```
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 19, 128)           82048
_________________________________________________________________
activation_2 (Activation)    (None, 19, 128)           0
_________________________________________________________________
dropout_2 (Dropout)          (None, 19, 128)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2432)              0
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 19464
_________________________________________________________________
activation_3 (Activation)    (None, 8)                 0
=================================================================
Total params: 102,280
Trainable params: 102,280
Non-trainable params: 0
_________________________________________________________________
```

Accuracy — 41.25%

Random Forest:

```
1
2  from sklearn.model_selection import GridSearchCV
3  #Creating the parameter grid based on the results of random search
4  param_grid = {
5      'bootstrap': [False],
6      'max_depth': [5, 7, 10],
7      'max_features': ['log2'],
8      'min_samples_leaf': [1, 2, 3],
9      'min_samples_split': [3, 5, 7],
10     'n_estimators': [2000, 1800, 1900, 2100]
11     }
12 rf = RandomForestClassifier()
13 #Instantiating the grid search model
14 grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
15                           cv = 3, n_jobs = 1, verbose = 2)
16
17 grid_search.fit(X_train, y_train)
```

APM_Audio.py hosted with ❤️ by GitHub view raw

We figured that given the insufficient amount of data we have to train a neural network properly, we can try out other models and see how they fare. The first one we used was good old Random Forest. It gave us an accuracy of 48%.

Hyper Parameter Tuning

The main challenge about training a random forest is the parameter tuning. With the sheer number of parameters, we first did a Randomized Search on the models. The way this works is that it basically picks random parameters from the options given and runs models based on that. We ran this for 50 iterations. This is a very useful method to narrow down the range of the parameters.

Based on the output of the best model from the Randomized Search, we found a smaller range of parameters to try out and performed a Grid Search on them using k-fold cross validation with $k = 3$.

This gave us the final model with the test accuracy of 48%, outperforming the Neural Network.

Support Vector Classifier:

For an SVC, we found that radial basis functions performed the best at emotion classification. Interestingly, using simple linear separation functions performed almost as well. Other SVC parameters (class weights, probability estimates) were not needed, and the analysis was conducted using a "one vs. rest" approach. SVC provided an accuracy of 51.25%.

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

mask = X['actor'] <= 20

x_train = X[mask].drop(['actor'], axis = 1)
y_train = y[mask]
x_test = X[-mask].drop(['actor'], axis = 1)
y_test = y[-mask]

x_train = StandardScaler().fit_transform(x_train)
x_test = StandardScaler().fit_transform(x_test) # scale train and test separately

svm = SVC(kernel = 'rbf', gamma = 'auto')
svm.fit(x_train,y_train)
```

Audio_SVC_XGB.py hosted with ❤ by GitHub view raw

XGBoost

With a vast number of parameters to choose for XGBoost, RandomizedSearchCV is once again useful in selecting a subset of 50 parameter combinations over which to cross-validate. This reduces training from many hours to just two or three. To our amusement/dismay, some of the selected parameters were very close to XGBoost's default values. But it didn't hurt to try, and the results surpassed the CNN and were close to par with the SVC. Accuracy of 50% was obtained.

```
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

# Too many parameters to try; use randomized search to pick a size 50 subset to test
params = {'subsample':[i/100 for i in range(25,76,10)],
          'eta':[i/10 for i in range(1,6)],
          'gamma':list(range(5)),
          'max_depth':list(range(4,9))}

clf = RandomizedSearchCV(XGBClassifier(), params, n_iter = 50, cv = 5, verbose = 2)
clf.fit(x_train,y_train)
print(clf.best_params_)

bst = XGBClassifier(eta = 0.1, gamma = 0, max_depth = 6, subsample = .35)
bst.fit(x_train,y_train)
```

Audio_SVC_XGB.py hosted with ❤ by GitHub view raw

. . .

Challenges and Future Scope

Combining audio & video data

This was undoubtedly our biggest challenge in this project. So far, we have split our data into separate audio and video files to extract MFCCs and images respectively. However, taking a combined approach to simultaneously train a model capable of processing audio and video signals would help achieve a more scalable outcome. As mentioned earlier in the blog, emotion recognition is majorly sought after in many industries. We believe as future scope, this product could be made more widely acceptable by being compatible with either kind of input.

Emotion recognition in Health Care:

An industry that's taking advantage of this technology currently is Health Care, with AI-powered recognition software helping to decide when patients necessitate medicine or to help physicians determine who to see first. A problem we foresee that can be prevented with accurate emotion detection is in the Mental Health awareness space. Those suffering from mental health issues often keep to themselves and don't share much about their problems. Correctly identifying emotions these distress signals, could make a huge difference to avoid mental breakdowns and stress-related trauma. A computer would be unbiased and more sensitive to detecting early signs to help alert close friends or family.

Dealing with the inherent bias

There are two broad biases that are suffered by our models:

1. All the actors are from the North American geographic area, and thus speak in a distinct North American accent, causing our models to be biased to that. Audio data from speakers of other geographic locations would help eliminate this bias.

2. All audio and video recordings are taken in a professional setting at Ryerson University in the absence of any background/white noise. Therefore, models that are trained on this dataset may not perform well on real-world data. A potential fix to this situation could be to train models on noisy audiovisual datasets and attach class labels using the Amazon Turk service.

References:

[1] Muneeb ul Hassan, VGG16 — Convolutional Network for Classification and Detection

[2] Sourish Dey, CNN application on structured data-Automated Feature Extraction

[3] Francesco Pochetti, Video Classification Experiments: combining Image with Audio features

[4] Ryan Thompson, How to Use Google Colaboratory for Video Processing

[5] James Lyons, Python Speech Features

[6] Angelica Perez, EmoPy: a machine learning toolkit for emotional expression

Machine Learning Data Science Neural Networks Emotions Artificial Intelligence

About Help Legal