Identifying Markers for Human Emotion in Breath Using Convolutional Autoencoders on Movie Data

Tom Février
Undergraduate student
Email: tomfevrier@gmail.com

Supervised by
Dr Patricia Riddle & Dr Jörg Wicker
Department of Computer Science
The University of Auckland
38 Princes Street, Auckland 1020, New Zealand

***Abstract*—This paper draws on research conducted by Williams *et al.*, from the Max Planck Institute for Chemistry and the Johannes Gutenberg University in Mainz (Germany). In 2013, they measured the concentrations of various volatile organic compounds (VOCs) emitted by the audiences in a cinema, in order to identify emotional responses to audiovisual stimuli. I propose a new approach based on deep learning to determine a correlation between the movie data and the content of the scenes, as well as the emotional response of the audiences—implied by the chemical composition of their breath. Three-dimensional convolutional neural networks, pre-trained on unlabelled data using a convolutional autoencoder, are used to predict labels and concentrations of VOCs from the pixels of the movies. Results are promising despite the small size of the datasets, and I suggest some ideas for future work to improve the accuracy of the models.**

***Index Terms*—Deep Learning, Movie Analysis, Emotional Response Analysis, Breath Analysis, Video Classification, Convolutional Neural Networks, Autoencoders**

I. Introduction

All living beings, and humans in particular, constantly emit chemicals into their environment, in their breath or through their skin. Williams *et al.* hypothesized that some of these volatile organic compounds (VOCs) could constitute markers for human emotion, as manifestations of instinctive biochemical responses [1]. In order to determine if audiences were emitting specific chemicals when faced with audiovisual stimuli, measurement instruments were installed in the ventilation system of a cinema in Mainz (Germany). Concentrations of carbon dioxide (CO_2) and hundreds of VOCs were collected every 30 seconds. The study involved close to 10,000 moviegoers, 108 screenings and 16 films, from December 1st, 2013 to January 14, 2014. Figure 1 shows the evolution of CO_2 concentrations over multiple time frames.

Williams *et al.* found that variations in the measurements were similar across different screenings of the same movie, except for intensities, which depend on the number of people in the audience. More interestingly, the variations were also similar for different chemicals: e.g. CO_2, acetone and isoprene, which have peaks occurring at the same times for a given movie.

Scene annotation was used to determine a causal link between the content of the films and the observed concentrations (see subsection II-A), however no automated approach to predict scene content and VOC measurements from the movie data (video and audio) had been attempted. My contribution consists in using deep convolutional neural networks on the movie pixels to perform a multi-label classification of the scenes and predict the emotional response of the audience. Some experiments were also conducted with audio data, using simple multi-layer neural networks.

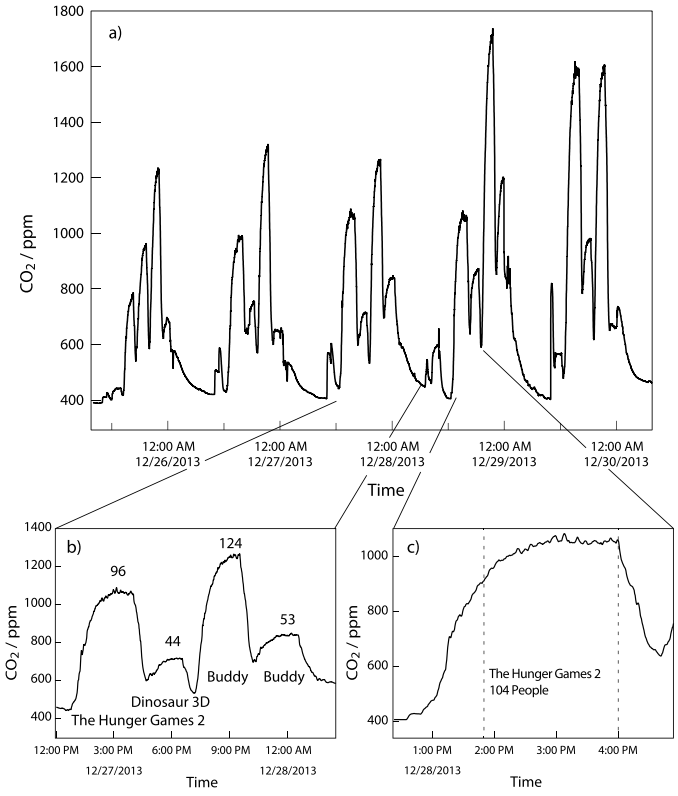


Fig. 1: **Selected sections of the CO_2 measurements** over (a) 5 days, (b) 1 day and (c) 1 film. The numbers above the peaks indicate the number of people in the audience. *From Williams* et al. [1]

II. Related work

A. Scene annotation and data mining

In order to find a correlation between the content of the films and the measured chemical concentrations, Wicker *et al.* used scene annotations [2]. Two types of labels were used: genre labels (`comedy`, `suspense`...) based on IMDb genre categories, and content labels (`conversation`, `action`...) describing what is happening in the scene. For some labels, sub-labels were used (e.g. `chase`, `hidden threat` and `hiding` within `suspense`). A list of all the labels and the ones selected for my study can be found in Table I.

The scene annotation was done manually by student volunteers from the Institute of Computer Science in Mainz (Germany), using a simple web interface [3]. The results were then averaged so that each 30-second period is associated with labels on which a majority of the annotators agreed.

Wicker *et al.* used three main data mining approaches. The first one is called *forward prediction*, and aims at predicting the current VOCs from the previous concentrations and labels—within a 5-minute time frame—, and from the current annotations. The second approach, *backward prediction*, assumes that the reactions of the audience occur *after* the corresponding scenes, and therefore uses the *future* VOCs to predict the current content of the scenes. The third approach combines the two previous algorithms, using the first model to predict future VOCs and the second model to predict the current features from the output of the first one. Hence, both models can be evaluated and improved: their performance was determined using the Area Under the Curve (AUC), which can be defined as the ratio between true positives—labels correctly predicted—and false positives.

They discovered a significant relationship between some of the labels and some of the chemicals, with AUC ranging from 0.76 to 0.85 for labels `injury` and `comedy`. Associated chemicals include methanol, which can be found in human breath, and quite surprisingly siloxanes, which are present in cosmetics and shampoo. The emission of siloxanes might be explained by emotionally induced variations in the body temperature, hence the causal link to `injury` scenes.

B. Convolutional neural networks

Convolutional neural networks (ConvNets) were introduced by LeCun *et al.* in 1989, in order to recognize handwritten zip codes on envelopes [4]. Biologically inspired by the architecture of our visual cortex, ConvNets are more suitable for image recognition than classic multilayer perceptrons, which do not scale well. ConvNets rely on two main features: *local connectivity* and *weight sharing*. Local connectivity implies that pixels within a small region of the image—known as a *receptive field*—are likely to form patterns, which filters can be trained to recognize. Weight sharing assumes that such patterns can be found in multiple locations in the image, and therefore that each filter can apply the same weights to every receptive fields.

A ConvNet architecture is mainly composed of two kinds of layers: convolutional layers and pooling layers. A convolutional layer is mainly defined by its number of filters and by its *kernel*, i.e. a small matrix of weights. For each image input, the convolution layer computes the dot product between the values of the pixels and the weights of each filter, by sweeping the kernel across the entire image (see Figure 2). It then generates for each filter a *feature map*, which can be fed into the next convolutional layer.

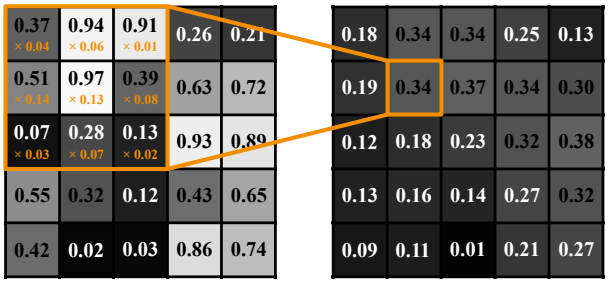


Fig. 2: **Convolution** on a 5×5 grayscale image, with a kernel size of 3×3. In this case, the convolutional layer assumes that pixels outside of the image are all zeroes when computing the edges of the feature map, a process known as *zero padding*. Only one convolution filter is shown.

Like any other neurons in classic neural network architectures, each "neuron" of the generated feature map needs to be activated. A common activation function for ConvNets is the *Rectified Linear Unit (ReLU)*, defined as $f(x) = max(0, x)$. In other words, ReLU simply changes all the negative values to 0. The main advantage of ReLU over other non-linear activation functions such as hyperbolic tangent ($tanh$) or sigmoid (defined as the inverse of $1 + e^{-x}$) is that it is much faster to compute [5].

Pooling layers are used to sample down the data, the most commonly used pooling function being *max pooling*. The max pooling layer partitions the image into "pools" and generates a smaller image by outputting the maximum value of each pool (see Figure 3). The size of the pool constitutes the factor by which the input image is scaled down.

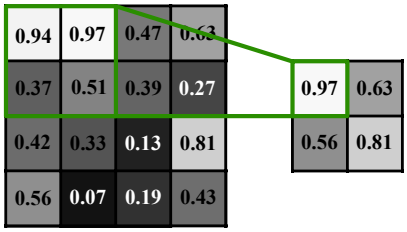


Fig. 3: **Max pooling** on a 4×4 grayscale image, with a pool size of 2×2.

Deep convolutional neural networks have been successfully used to classify images for the past few years, achieving performances close to those of humans on the classic *ImageNet* database [6]. However, little research has been conducted on video classification. In 2014, Tran *et al.* were able to extract spatio-temporal features from video clips using three-dimensional convolutional networks [7], a technique usually applied to 3D objects recognition [8] or segmentation of volumetric medical imaging [9].

The idea behind this method is to preserve the temporal information of the input video by performing convolutions over a volume composed of multiple frames, using the third dimension not as a spatial one but rather as a temporal one. 3D convolution and max pooling layers behave in a similar fashion to their 2D counterparts, except kernels and pools are three-dimensional.

Tran *et al.* found that for video classification, 3D ConvNets outperform 2D ConvNets applied to each frame individually. They also proved empirically that a $3 \times 3 \times 3$ convolution kernel seemed like an optimum for low resolution videos [7].

C. Autoencoders

The main drawback to convolutional neural networks is that they require a significant amount of training data in order to be efficient, since features are slowly learnt through backpropagation. However, real world data is usually scarce and the results are highly dependent on the initialization value of the weights. One way to circumvent this problem is to use autoencoders, which allow ConvNets to be pre-trained on unlabelled data [10].

Autoencoders are made of two parts: the encoder and the decoder. The encoder learns to compress the input data into a short code, while the decoder learns to uncompress that code in order to reconstruct the original data (see Figure 4). After training the autoencoder on unlabelled data for a long time (with the same dataset as input and output), we can assume that the code is a good representation of the original data, and that it can be used as the input layer of another multi-layer network.

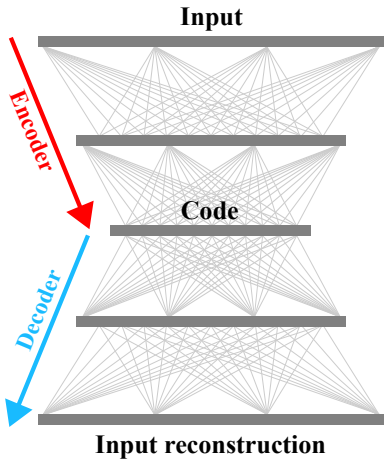
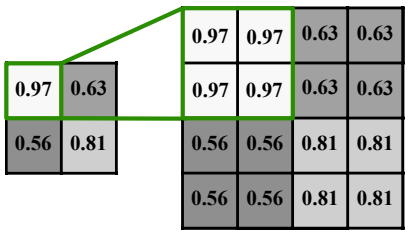
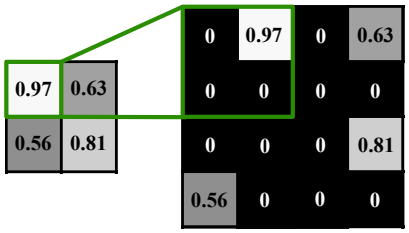


Fig. 4: **Basic multi-layer autoencoder.**

Convolutional autoencoders can be built by stacking a convolutional network and an inverse convolutional network on top of each other. Inverse ConvNets are composed of two types of layers: *deconvolutional layers* and *unpooling layers*. Deconvolutional layers, unless we want them to use the same weights as their encoding counterparts—such layers are known as "transposed convolutional layers" [10]—, are virtually the same as convolutional layers. However, pooling being a non-invertible operation, unpooling layers have to use approximations to reconstruct the data. The most simple one is the nearest-neighbour interpolation, which consists in repeating the data for each pixel of the pool, but unpoooling layers can rather keep track of the location of the maximum values using "switches" and set the other locations to zero [11]. The difference between these two upsampling methods can be seen in Figure 5. One of the advantages of using switches is to preserve the original structure of the image, whereas with the nearest-neighbour method details are lost and the image tends to be blurry.



(a) Unpooling using nearest-neighbour interpolation



(b) Unpooling using switches

Fig. 5: **Unpooling** on a 2×2 grayscale subsample, with a pool size of 2×2. The original image is the one from Figure 3.

D. Batch normalization

Training deep neural networks can take a long time, as the distribution of the inputs of each layer changes over time when weights are updated. To overcome this issue, Ioffe and Szegedy introduced in 2015 a method known as batch normalization [12]. For each batch, the output of the previous layer is normalized by subtracting the batch mean and dividing by the batch standard deviation. However, by changing the data representation, the weights in the next layer are no longer optimal. In order to fix this, two trainable parameters are introduced: the output of the normalization layer is multiplied by a "standard deviation" γ and shifted by a "mean" β. This way, backpropagation can "undo" the normalization if it helps minimising the loss.

For each value x_i within a batch $\mathcal{B} = \{x_1, ..., m\}$, where y_i is the result of batch normalization applied to x_i:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \textit{(mean)}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \textit{(standard deviation)}$$

$$\hat{x_i} \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}}}} \qquad \textit{(normalization)}$$

$$y_i \leftarrow \gamma\hat{x_i} + \beta \qquad \textit{(scale and shift)}$$

In addition to significantly accelerating the training, batch normalization also helps prevent overfitting, i.e. a situation in which the model fits too well to the training data, and therefore does not have any predicting power on data it was not trained with. Some techniques can be used to prevent overfitting, such as dropout [13], which consists in randomly dropping a fraction of the input data during training, by setting their values to zero. However, Ioffe and Szegedy found that batch normalization produces the same effect as dropout [12]: since the values are normalized they do not depend on the training examples anymore, and thus the network can generalize well.

III. METHODS

A. Datasets and pre-processing

This study focus on the same six movies as the ones studied by Wicker *et al.* [2]:

- ***Buddy*** (Michael Herbig, 2013)
  Abbreviated as `Buddy`, 168 sequences.
- ***The Hobbit: The Desolation of Smaug*** (Peter Jackson, 2013)
  Abbreviated as `Hobbit`, 289 sequences.
- ***Machete Kills*** (Robert Rodriguez, 2013)
  Abbreviated as `Machete`, 197 sequences.
- ***The Secret Life of Walter Mitty*** (Ben Stiller, 2013)
  Abbreviated as `Mitty`, 200 sequences.
- ***Paranormal Activity: The Marked Ones*** (Christopher B. Landon, 2014)
  Abbreviated as `Paranormal`, 150 sequences.
- ***The Hunger Games: Catching Fire*** (Francis Lawrence, 2013)
  Abbreviated as `Tribute`, 271 sequences.

a) Video: The video data was processed using the ffmpeg library [14], by extracting one frame per second for each movie. The resulting images were too large to be fed into the convolutional neural network, so they were resized to reasonable dimensions using the ImageMagick library [15]. 64×36 pixels seemed like a good compromise between performance and quality, given the available hardware: details such as facial expressions are lost, but the structure and dynamics of the shot are preserved. If the aspect ratio of the movie was different (usually 1.85:1 or 2.39:1), the images were slightly distorted to fit into these dimensions. Frames corresponding to the closing credits were dropped, as they are irrelevant in this study. Finally, the frames were bundled together for each 30-second sequence (30 frames), and the pixels were read and converted into a `(nb_sequences, 30, 36, 64, 3)`-sized numpy matrix using the OpenCV Python library [16].

b) Labels: The labels dataset was introduced by Wicker *et al.* [2]. Each movie corresponds to a single CSV (Comma Separated Values) file, with the columns describing which of the 42 labels were used to describe each 30-second sequence of the film, using boolean values (`0` or `1`). The first thing I had to do was to transpose the data, so that the columns matched the labels rather than the sequences. The labels that were applied to less than 20 sequences (out of 1274) were assumed to be irrelevant, and were dropped. Table I shows a list of all labels and the 13 that were kept.

c) VOCs: The VOCs dataset was also introduced by Wicker *et al.* [2]. For a given screening, the measurements of VOCs—identified by their molecular mass—for every 30-second periods are contained in a single ARFF (Attribute-Relation File Format) file. This format, based on CSV, was developed for use within the University of Waikato's machine learning software Weka. All I had to do was to skip the header containing the names of all the attributes and their types, and select the columns corresponding to chemicals of interest: CO_2, ethanol (m 46.0653 or 46.07 g·mol^{-1}), methanol (m 33.0335), acetone (m 58.0756), isoprene (m 70.0766) and siloxane (m 374.0818).

CO_2 and methanol are found in breath, while ethanol corresponds to alcohol consumption in the cinema—a useful information to understand the emotional state of the audience. Acetone is a gas linked to fat catabolism, and isoprene plays a role in cholesterol synthesis and has been identified as a result of breath holding and twitching muscles [1]. Finally, siloxane, found in cosmetics, could be an indicator of body temperature variations (see subsection II-A).

Labels	Sub-labels
suspense	chase hidden threat hiding
comedy	romantic comedy
crime	courtroom drama detective murder scene
fantasy	
horror	
mystery	
romance	
drama	
everyday life	
dream	
landscape	
conversation	aggressive conversation calm conversation **conversation main actor** intrigue
action	
death	many deaths main character dies villain death
running	
recovery	
hero recovers	
laughter	
sleeping	
blood (violence)	
sex	
kissing	
crying	main character cries other character cries
pets	
injury	
sudden shock	

TABLE I: **Labels used for scene annotations** by Wicker *et al.* [2] The ones selected for classification are in bold.

The data was then normalized between 0 and 1 by applying the following formula to each screening and each chemical:

$$vocs = \frac{vocs - voc\_min}{voc\_max - voc\_min}$$

However, for some screenings the data was non-existent. In such cases, I had no choice but to set the missing values to 0.5: this is not an optimal solution, but given the scarcity of the dataset I could not afford to sacrifice an entire screening just because the values for one single chemical were missing. Besides, this problem only affects 0 to 4 screenings out of 46 for each chemical.

d) Audio: Although my study focuses mainly on video data, I also conducted some experiments with audio data. The video files were processed using the ffmpeg library [14], resulting in one audio clip for each 30-second sequence of each movie. These files were then analysed using the Librosa library [17], and some features of interest were extracted: mel-frequency cepstral coefficients (MFCCs), mel-scaled spectrogram, spectral contrast, chromagram and tonnetz.

I will not go into too much details here, but MFCCs can be obtained by taking the Fourier transform (see Figure 6) of an audio signal, mapping it onto the mel scale—a frequency scale commonly used because it approximates the human perception

of sound [18]—, and then applying a linear cosine transform on the logs of the powers for each resulting frequency.

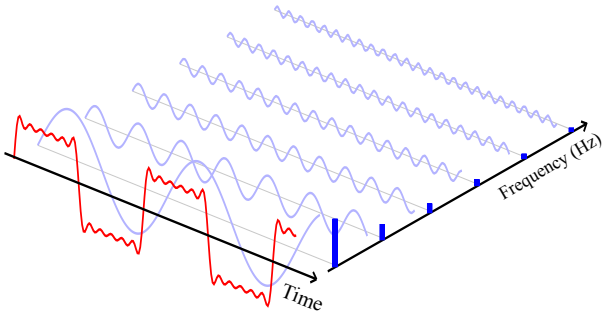


Fig. 6: **Fourier transform.** The audio signal is decomposed into sine waves, and the amplitude for each frequency is plotted on a second graph.

A spectrogram is a visual representation of an audio signal: the x axis is time, the y axis is frequency (on the mel scale here), and colours represent the amplitude, usually in decibels (dB). Chromagram is quite similar, except that the y axis corresponds to the 12 pitches of the equal-tempered scale: C, C#, D, D#, E ,F, F#, G, G#, A, A#, B. Finally, tonnetz uses a different scale, based on the classic theory of fifths, major thirds and minor thirds [19]. Figure 7 displays some of these features.

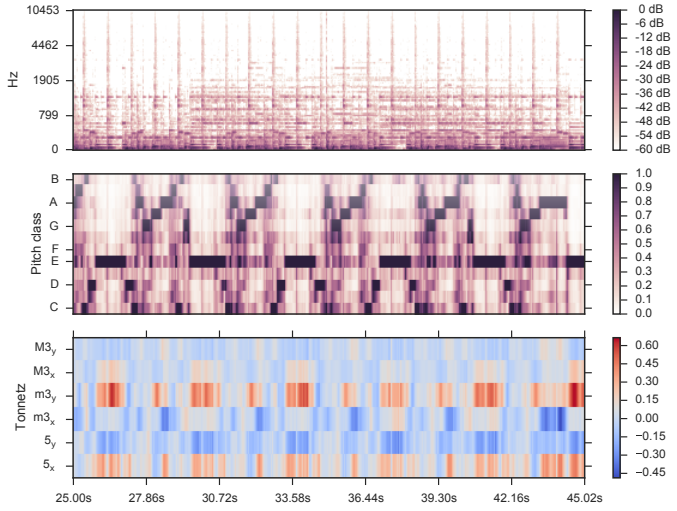


Fig. 7: **Examples of features extracted** on a 20-second audio clip [18]. From top to bottom: mel-scaled spectrogram, chromagram and tonnetz.

The extracted features are then averaged and converted into a `(nb_sequences, 193)`-sized numpy matrix (40 values for MFCCs, 128 for the mel-scaled spectrogram, 7 for spectral contrast, 12 for the chromagram, and 6 for tonnetz).

B. Autoencoder

The neural networks were implemented using the Keras [20] library, using Google's Tensorflow as a backend. Keras has the advantage of providing a high level API in Python, with core layers (fully-connected, convolutional, pooling, batch normalization, etc.), loss functions and optimizers already implemented as ready-to-use modules.

Given the limited size of the datasets, a convolutional autoencoder was used to pre-train the model on unlabelled data. The dataset was extended with six new films, selected either for their similarity with the studied movies, or because they seemed representative of a given genre. The video data was processed in the same way as that described in subsection III-A. These movies are:

- ***The Lord of the Rings: The Return of the King*** (Peter Jackson, 2003, extended edition)
  Chosen for its similarity to *The Hobbit: The Desolation of Smaug* and for its duration (263 minutes!).
  Abbreviated as `Sauron`, 482 sequences.
- ***Machete*** (Robert Rodriguez, 2010)
  First instalment of the franchise.
  Abbreviated as `Machete_bis`, 201 sequences.
- ***Hector and the Search for Happiness*** (Peter Chelsom, 2014)
  Chosen for its similarity to *The Secret Life of Walter Mitty*.
  Abbreviated as `Hector`, 228 sequences.
- ***Paranormal Activity*** (Oren Peli, 2007)
  First instalment of the franchise.
  Abbreviated as `Paranormal_bis`, 169 sequences.
- ***The Hunger Games*** (Gary Ross, 2012)
  First instalment of the franchise.
  Abbreviated as `Tribute_bis`, 266 sequences.
- ***Furious 7*** (James Wan, 2015)
  Chosen as an example of an action film.
  Abbreviated as `Furious`, 262 sequences.

The architecture of the three-dimensional convolutional autoencoder can be seen in Figure 8. The encoder is composed of the following layers:

- 3D convolutional layer (`Conv3D`) with 32 filters and a kernel size of $3 \times 3 \times 3$
- 3D max pooling layer (`MaxPooling3D`) with a pool size of $2 \times 2 \times 2$
- 3D convolutional layer with 64 filters and a kernel size of $3 \times 3 \times 3$
- 3D max pooling layer with a pool size of $2 \times 2 \times 2$

This kind of architecture—alternation of convolutional and pooling layers—is common for convolutional neural networks, while increasing the number of filters in deeper layers allows the network to detect new patterns as the representations get more compressed and abstract.

The resulting encoded representation is a volume of $64 \times 16 \times 9 \times 7$ values. The decoder is the inverse operation: two deconvolutional layers with the same kernel sizes and numbers of filters as their encoding counterparts, each of them followed by an unpooling layer. Since Keras does not support "switch" unpooling (yet), the nearest-neighbour interpolation (called `UpSampling` in Keras) is used. All convolutional and deconvolutional layers use zero padding (see Figure 2, known as `same` in Keras) so that the shape of the volume is preserved.

However, since max pooling performs an integer division on all dimensions, the depth of the resulting volume is 2 pixels shorter than the original volume. To solve this problem, zero padding (`ZeroPadding3D`) is applied to the z axis, i.e. two black frames are added at the front and at the back of the volume. Finally, another deconvolution using the sigmoid activation function (see below) is performed on the volume: this way, the values of our reconstruction are in the [0, 1] range as they should be. All the other activation functions are ReLUs. In order to accelerate training and prevent overfitting,

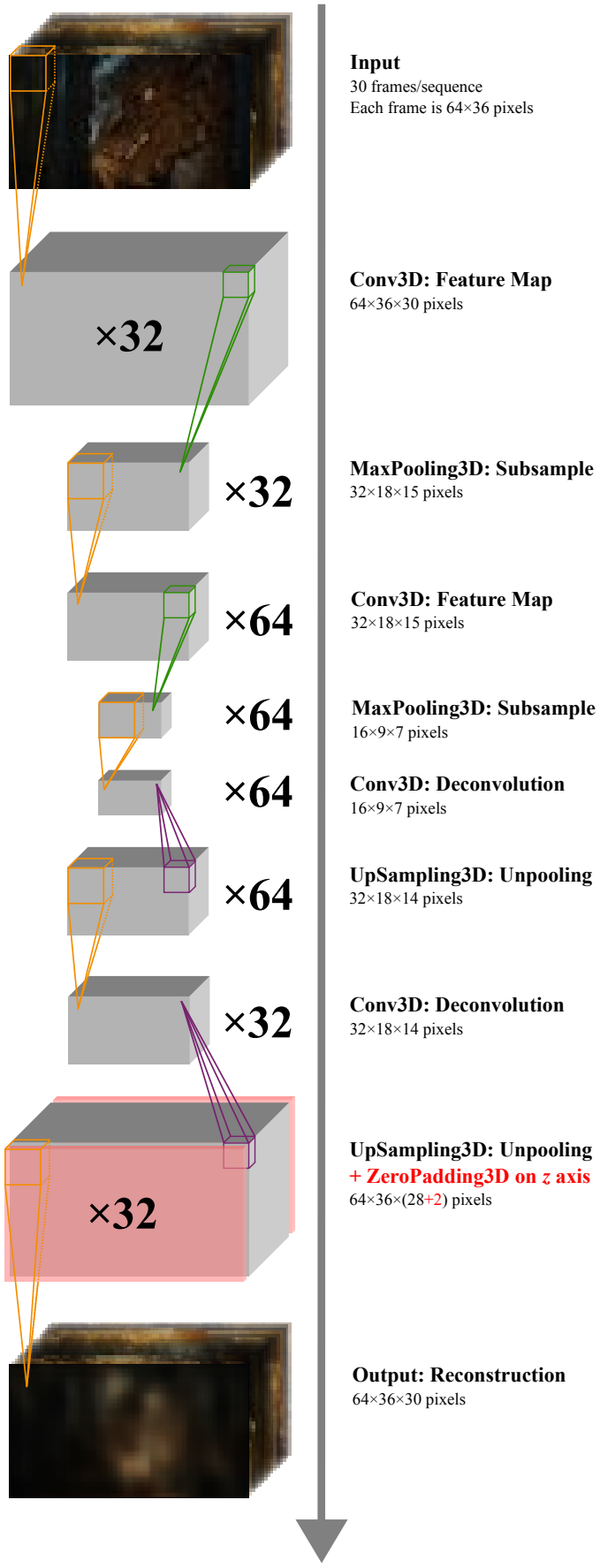


Fig. 8: **Architecture of the 3D Convolutional Autoencoder.** Convolutions and deconvolutions (orange) have a kernel of size $3 \times 3 \times 3$, max pooling layers (green) and unpooling layers (purple) have a pool size of $2 \times 2 \times 2$. Batch normalization layers and activation functions are not shown.

batch normalization layers are added after each convolutional layer.

The autoencoder was trained over 50 epochs using Keras' `binary_crossentropy` loss function and `Adadelta` optimizer [21], which controls the learning rate and decay of the model. The first 80% of each movie are used as a training set, and the last 20% for validation. The training set contains 2301 samples and the validation set 582 samples. Samples are not shuffled before training, as it is assumed that the network can learn from the sequential properties of each movie.

I was granted access to a computer in the IVS lab of the University of Auckland, with the following specifications:

- CPU: Intel Core i7-4820K, 4 cores, 3.70 GHz
- GPU: NVIDIA GeForce GTX 580
- RAM: 48 GB

Using GPU acceleration, training took around 46 seconds per epoch, and 51 seconds per epoch with batch normalization. The total number of trainable weights is 227,331: 2624 for the first convolution, 55,360 for the second one, 110,656 for the first deconvolution, 55,328 for the second one, 2595 for the final one, and the remaining weights are the parameters of the batch normalization layers. By default, the weights of the convolutional layers are initialized using a Glorot uniform distribution, and the β and γ parameters are initialized to 0 and 1 respectively.

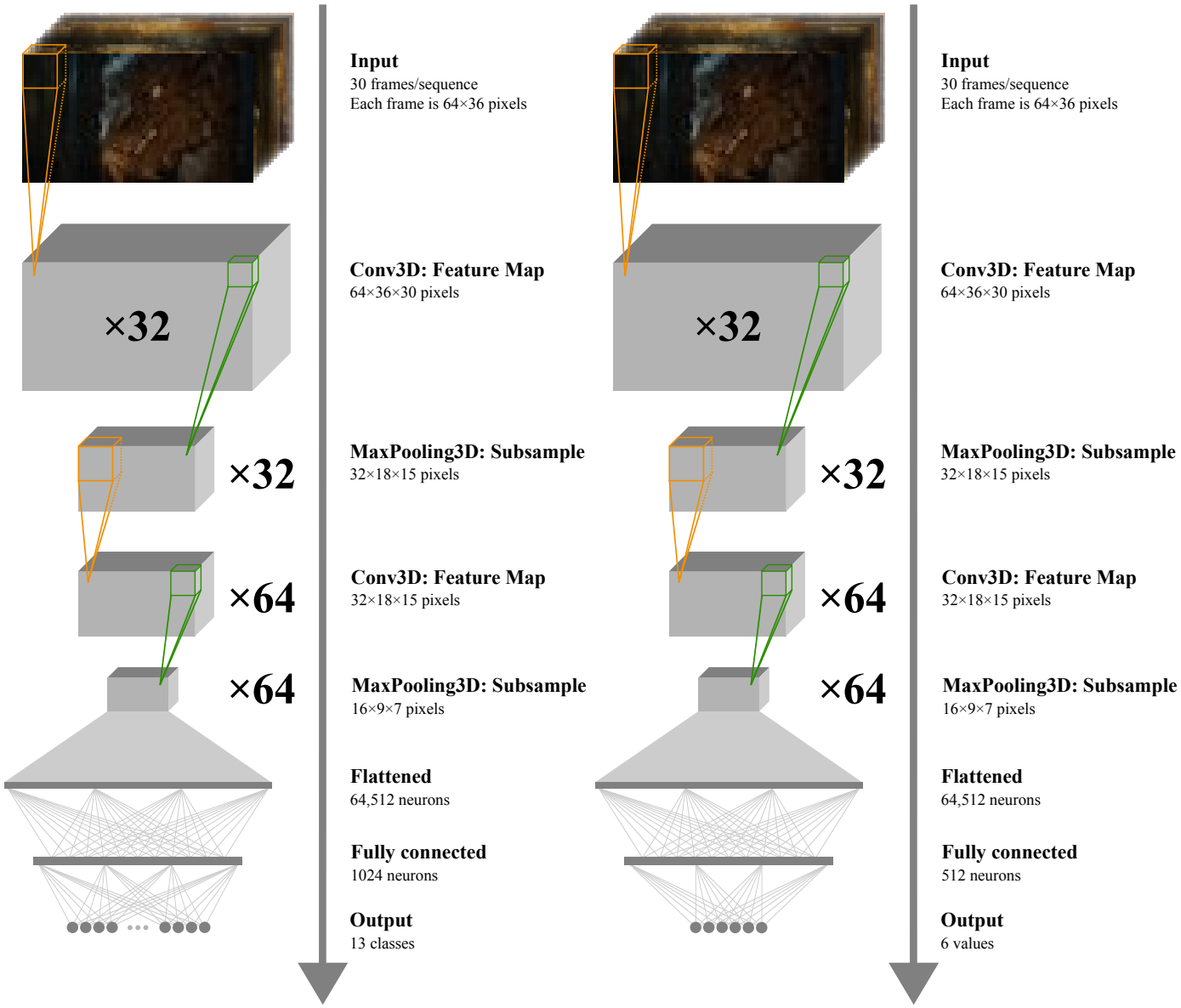
After training, the layers of the decoder are dropped, the weights of the convolutional and batch normalization layers are frozen, and the encoded representation is flattened (i.e. converted from a volume to a vector). Then, fully-connected layers related to the task (either classification or prediction) are appended.

C. *Multi-label classification*

The classification network (see Figure 9-a) is composed of two fully-connected layers. The last layer is the output layer: each of the 13 neurons corresponds to one label, and is activated using a sigmoid function. Indeed, since sigmoid outputs a probability between 0 and 1, it is particularly appropriate for multi-label classification as it provides the percentage on how likely a label applies to the input. In practice, this value is rounded to either 0 or 1 before evaluating the model: it is assumed that a probability over 0.5 means that the model considers this label relevant.

I chose to use 1024 neurons for the ReLU-activated intermediary layer, so that the number of neurons is divided by roughly the same factor between the previous layer (64,512 neurons) and the next one (13 neurons). Batch normalization is also added to the output of this layer, in order to fix potential weight initialization problems due to the concatenation of the autoencoder and fully-connected classifier.

The classification network was trained over 100 epochs using Keras' `binary_crossentropy` loss function and `Adadelta` optimizer. As with the autoencoder, the first 80% of each movie are used as a training set, and the last 20% for validation. The training set contains 1017 samples and the validation set 257 samples. Using GPU acceleration, training took around 11 seconds per epoch. The total number of trainable weights is 66,076,685: 66,061,312 for the intermediary layer, 13,325 for the output layer, and the remaining weights are the parameters of the batch normalization layer.



(a) Multi-label classification 3D ConvNet.

(b) VOC prediction 3D ConvNet.

Fig. 9: **Architectures of the classification and prediction 3D ConvNets.** Convolutions (orange) have a kernel of size $3 \times 3 \times 3$, max pooling layers (green) have a pool size of $2 \times 2 \times 2$. The weights of the convolutional layers are frozen. Batch normalization layers and activation functions are not shown.

D. VOC prediction

The prediction network (see Figure 9-b) is also composed of two fully-connected layers. The last layer is the output layer: the 6 neurons correspond to the predicted concentrations of the chemicals of interest. Since concentrations are discrete values and not probabilities, no activation function is used. I chose to use 512 neurons for the ReLU-activated intermediary layer for the same reason as above, and batch normalization is also added.

The prediction network was trained over 100 epochs using Keras' `mean_absolute_error` loss function and `Adadelta` optimizer. Since multiple screenings are available for each movie, the validation set is constituted in two different ways: by using the last 20% of all movies, as with the autoencoder and classification network, or by using one screening for each film. However this structure creates disparities between movies, as there are only a few screenings for some films like `Machete` or `Paranormal` (2 and 4 respectively), whereas other films were projected up to 14 times (for `Mitty`). Leaving one movie out for validation was also attempted, but the model does not seem general enough (yet) to be able to predict values for a film it was not trained on at all.

Using the last 20% for validation, the training set contains 7933 samples and the validation set 2001 samples. When leaving one screening out for validation, the training set contains 8860 samples and the validation set 1274 samples. Using GPU acceleration and batch normalization, training took around 62 seconds per epoch when using the last 20%, and around 67 seconds per epoch when leaving one screening out. The total number of trainable weights is 33,034,758: 33,030,656 for the intermediary layer, 3078 for the output layer, and the remaining weights are the parameters of the batch normalization layer.

E. Using audio data

The classification and prediction models for the audio data are simple multi-layer neural networks. The first layer corresponds to the input features (193 neurons), then two hidden layers (of size 280 and 300 respectively) are appended in order to broaden the data. The last layer is the output layer, either composed of 13 sigmoid-activated neurons or 6 non-activated neurons depending on the task.

The classification network was trained over 1000 epochs using Keras' `binary_crossentropy` loss function and `Adadelta` optimizer. The last 20% of the samples are used for validation. The training set contains 1039 samples and the validation set 263 samples. Using GPU acceleration, training took less that 0.1 second per epoch. The total number of trainable weights is 142,533: 54,320 for the first hidden layer, 84,300 for the second one, and 3913 for the output layer.

The prediction network was trained over 1000 epochs using Keras' `mean_absolute_error` loss function and `Adadelta` optimizer. Using the last 20% for validation, the training set contains 7969 samples and the validation set 2026 samples. When leaving one screening out for validation, the training set contains 8714 samples and the validation set 1281 samples. Using GPU acceleration, training took around 1 second per epoch, regardless of the structure chosen for the validation set. The total number of trainable weights is 140,426: 54,320 for the first hidden layer, 84,300 for the second one, and 1806 for the output layer.

IV. Results

A. Advantages of batch normalization

As expected, batch normalization significantly accelerates the training of the autoencoder, as demonstrated in Figure 10. The training and validation losses converge much more quickly with batch normalization, and the trends are smoother, whereas validation loss fluctuates a lot without batch normalization. After 50 epochs, training loss is about the same for both implementations (0.4305 and 0.4299), however validation loss is significantly lower with batch normalization (0.4145 vs 0.4196), which indicates that it helped prevent overfitting.

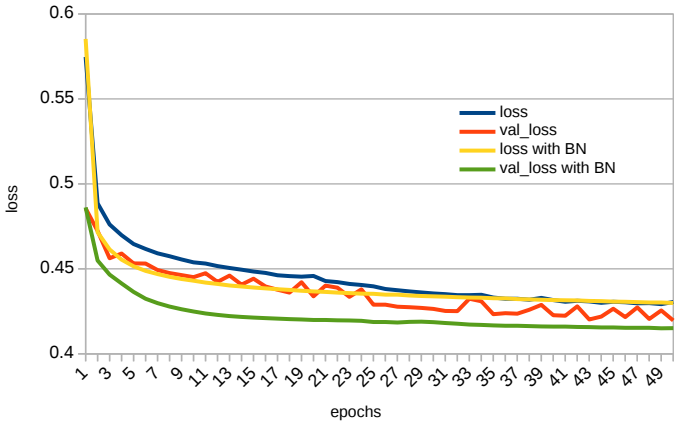


Fig. 10: **Evolution of the loss and validation loss of the autoencoder**, with and without batch normalization, over 50 epochs.

Figure 11 demonstrates the effect of batch normalization on the quality of the reconstruction. With batch normalization, the colours are closer to those of the original shots, and the reconstructions seem sharper as well. This can be explained by the fact that without batch normalization the autoencoder tries to match the input from randomly initialized weights, leading to "greyish" reconstructions that get more and more colourful as the training progresses. With batch normalization, the data is spread out at each step, so it is easier for the network to converge towards the wanted colours. The improvement in the training speed is also visible, as reconstructions after 10 epochs with batch normalization seem almost as good as reconstructions after 50 epochs without it.

B. Performance of the multi-label classification

Keras' loss value does not provide any insights regarding the performance of the classification for each label. In order to assess the accuracy of the neural networks, I used the *precision*, *recall* and F_1 *score*. Precision can be defined as the proportion of relevant labels among those selected, whereas recall corresponds to the proportion of selected labels among those relevant. In other words:

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

In this case, true positives are labels correctly predicted as relevant, false positives are labels incorrectly predicted as relevant, and false negatives are labels incorrectly predicted as irrelevant.

The F_1 score is defined as the harmonic mean of the precision and recall values:

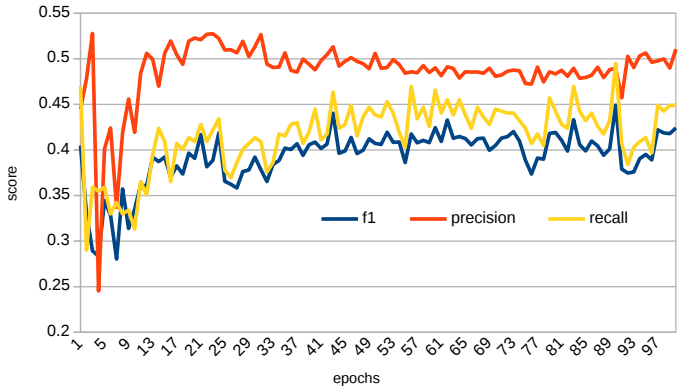
$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

$$= 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

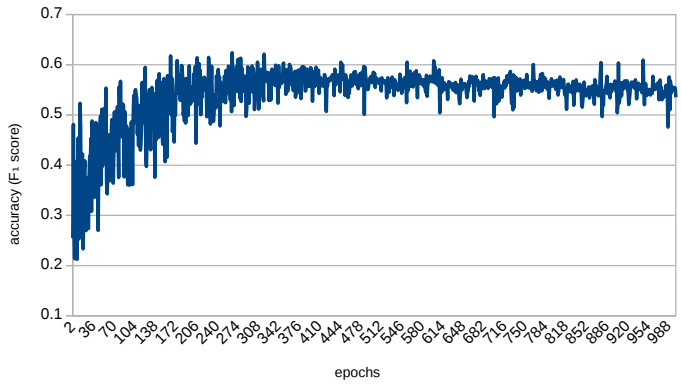
Here, since some labels are much more used than others, the averages of the precision, recall and F_1 score are weighted by the number of times each label is applied to samples from the validation set.

The results when using the video data as input can be seen in Figure 12-a. The accuracy—expressed by the F_1 score—varies a lot but seems to stabilize around 42% after 100 epochs, with a precision of 51% and a recall of 45%. Among the labels, only four of them have a non-zero accuracy: `conversation` (73%), `suspense` (43%, but with a precision of 81%), `conversation main actor` (39%) and `action` (14%). The high performance of `conversation` can be explained by the high support for this label, which was applied to 149 validation samples out of 257.

The results for the audio data are displayed in Figure 12-b. The accuracy fluctuates a lot, but seems to level off at around 55%, with a precision rate of 63% and a recall rate of 50%. The labels with the highest accuracy are `action` (66%), `conversation` (64%) and `suspense` (63%). These labels correspond to situations with specific types of sounds, which could explain why they perform so well. Overall, the audio data seems to be more insightful than the video data to classify the content of the scenes.

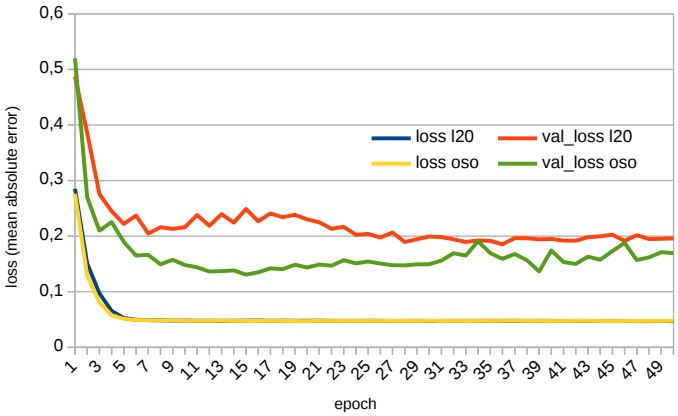


(a) Evolution of the F_1 score, precision and recall on video data, over 100 epochs.

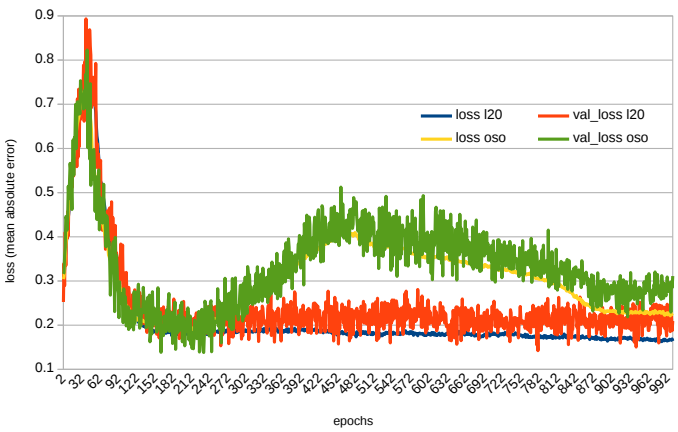


(b) Evolution of the F_1 score on audio data, over 1000 epochs.

Fig. 12: **Evolution of the performance of the video and audio classification networks**



(a) Evolution of the loss and validation loss on video data, over 50 epochs.



(b) Evolution of the loss and validation loss on audio data, over 1000 epochs.

Fig. 13: **Performance of the video and audio prediction networks**, using either the last 20% of each movie (`l20`) or one screening out (`oso`) for validation.

C. Performance of VOC predictions

The performance of the prediction networks was evaluated using the mean absolute error. Let y_{targ} be the output value in the validation set and y_{pred} the value predicted by the network, the mean absolute error is defined as:

$$MAE = \frac{\sum_{i=1}^{n} | y_{pred} - y_{targ} |}{n}$$

The results of the predictions using video data are displayed in Figure 13-a, when using either the last 20% of each movie or one screening out for validation. In both cases, the loss seems to stall after 10 epochs, which is a sign that the model has achieved his best performance. However the predictions are mediocre, with a validation loss levelling off at around 0.19 when using the last 20%, and 0.15 when leaving one screening out. In the latter case, the model seems to be experiencing overfitting after 30 epochs, as the validation loss starts fluctuating a lot.

The results for the audio model can be seen in Figure 13-b. Overall the accuracy is very poor, but using the last 20% of each movie for validation seems to perform better, with a validation loss at around 0.21 (compared to 0.28 when leaving one screening out for validation). However validation loss varies a lot, so although a clear trend is visible it is difficult to give any specific value.

D. Visualization of features learnt by the network

The downside of deep neural networks is that it is very difficult to visualize what networks actually learn. For convolutional neural networks though, it is possible to visualize what kind of features they were trained to identify. Activation maps and subsamples can be generated using Keras, by predicting the output of each convolutional or pooling layer. Figure 14 displays some of these outputs as heat maps.

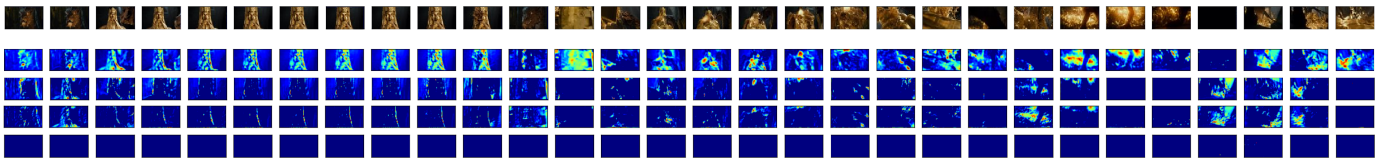
On Figure 14-a, four examples of activation maps are presented. The first filter seems to be sensitive to bright colours, as the activated zones correspond to the golden statue of the dwarf king and explosions. The second one, surprisingly, appears to be activated by the background but not by the main elements of the scene. The third filter is quite remarkable, as it seems to be detecting edges and strong changes from one frame to another. The last filter is an example of a "useless" filter, as it does not activate at all. It is interesting to notice that since the convolution kernel has a depth of 3 frames, the filters seem to "anticipate" images: e.g. for the second and third filter, the second frame already represents the statue, even though the original frame still depicts the dragon.

As we go deeper into the network, it becomes more and more difficult to identify what the outputs actually represent, especially with the pooling layers compressing the data temporally. The encoded representations (Figure 14.d) have very little in common with the original frames, but the decoder manages to reconstruct the latter anyway.

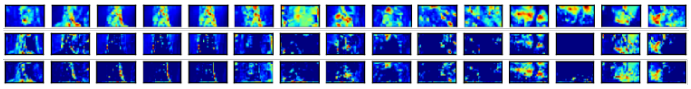
(a) Without batch normalization. *The random initialization of the weights explains the "greyish" look of the reconstruction for the first few epochs.*

(b) With batch normalization. *The normalized data is more spread out, hence these surprisingly vibrant colours for the first few epochs.*

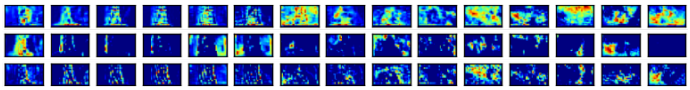
Fig. 11: **Reconstruction of the input data** for one random sequence for each movie. Only the first frame of each sequence is shown, though the data is three-dimensional. Original frames are visible on top, then from top to bottom for (a) and (b): reconstruction after 1 epoch, reconstruction after 10 epochs, reconstruction after 50 epochs.



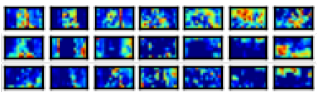
(a) Activation maps of some of the filters after the first convolutional layer. $64 \times 36 \times 30$ pixels.



(b) Same activation maps after max pooling. $32 \times 18 \times 15$ pixels.



(c) Activation maps of some of the filters after the second convolutional layer. $32 \times 18 \times 15$ pixels.



(d) Same activation maps after max pooling. $16 \times 9 \times 7$ pixels.

Fig. 14: **Visualization of the outputs of the encoding layers of the 3D Convolutional Autoencoder** when applied to one sequence from `Hobbit`. Original frames are visible on top. Colours indicate the level of activation, from dark blue to red.

V. Discussion and Future Work

In my research, I had to face the limitations of the original data. Among the selected labels, only 10 had been applied to samples from the validation set, and only 4 had been applied to more than 10% of these samples: `conversation` and `conversation main actor` (58% and 29.2% respectively), `suspense` (52.5%) and `action` (22.6%). The classification network performs moderately with these categories, but the overall performance suffers from the lack of support for most labels. Regarding the VOC dataset, some values were missing and had to be dealt with in a really "hacky" way (see subsection III-A): instead of replacing the missing values with 0.5, maybe it would be better to use the mean for this chemical on the other screenings of the same movie. Besides, it is unsure whether the movies have an immediate observable effect on the emission of chemicals by the audience, or whether this response is observed a few minutes after the scene that triggered it.

The rather poor results of the convolutional neural networks presented in this paper can also be explained by the small size of the datasets: 1274 samples for the labels, 10,134 samples for the VOCs, and 2883 samples for the autoencoder. This is not nearly enough to achieve significant results and to obtain

a model that can generalize well. To circumvent this problem, one could use data augmentation to artificially increase the size of the datasets. Krizhevsky *et al.* explored some techniques to augment training data [22]. By slightly zooming into images and adding random translations and horizontal reflections, they were able to increase the size of their training set by a factor of 2048. Data can also be augmented by slightly altering the RGB values of the pixels: such changes are barely visible to the human eye, but to the neural network it seems like a completely different image.

The performance of the convolutional autoencoder could be improved as well, by training it on more movies. Since data does not have to be labelled, any films can be used, and it is safe to assume that the more movies the autoencoder is trained on, the more generalized the encoded representation should be. Also, using switch unpooling instead of nearest-neighbour interpolation (see subsection II-C) in the decoder could enhance the quality of the encoded representation as well. This seems counter-intuitive, but since the direction and intensity of backpropagation depend on the accuracy of the reconstruction, preserving the structure of the original data in the decoder actually helps improve the encoder [11].

For this study, I was also limited by the hardware I was using. I had no choice but to use low resolution frames (64×36 pixels), as larger images (128×72 pixels for instance) could not fit into memory, and training would have taken too much time anyway. With more memory and a more powerful GPU—or multiple GPUs—, one could apply this network to higher resolution frames, allowing convolutional layers to detect more details such as facial expressions.

Concerning audio data, instead of extracting numerical values from mel-scaled spectrograms and chromagrams like I did, two-dimensional convolutional networks can be applied directly to these visual representations of audio signals. Piczak found that this technique, although more difficult to implement, achieves similar or better results compared to other models using neural networks on audio features, with accuracies ranging from 64.5% to 80% on multiple datasets of environmental recordings [5]. As a matter of fact, local connectivity—the fundamental principle of convolutional neural networks—applies perfectly to representations of audio signals, as segments following each other in time and close frequencies are likely to form patterns.

Along with video and audio data, dialogues could also be used for sentiment analysis on movie scenes. Tang *et al.* used a combination of convolutional networks and long short term memory (LSTM) networks on text samples, in order to classify IMDb and Yelp reviews [23]. One could try to apply such methods to movie subtitles to predict the emotions conveyed by a given scene.

All the datasets, files and trained models are available online at https://github.com/TomFevrier/smelloffear-deeplearning.

VI. Conclusion

In this paper I presented a new approach, based on three-dimensional convolutional neural networks, to predict the content of movie scenes and the emotional response of the audience from the pixels of the films. To circumvent data scarcity, the model was pre-trained on unlabelled data using a convolutional autoencoder, and batch normalization was used to prevent overfitting and reduce training time. After training, the models were able to classify the content of the scenes with accuracies ranging from 14% to 73%, and to predict the concentrations of volatile organic compounds (VOCs) with error rates between 0.15 and 0.19. Experiments on audio data were also conducted, by extracting various features from the audio signals and feeding them into a simple multi-layer neural network: accuracies up to 66% were achieved for the multi-label classification, and VOC concentrations were predicted with a mean absolute error of 0.21. However there is room for improvement, as these datasets hold great potential for future work with deep learning.

VII. Acknowledgements

I would like to thank my supervisors Dr Patricia Riddle and Dr Jörg Wicker for their guidance, as well as Dr Patrice Delmas for granting me access to the IVS lab, and Alex Peng for his helpful advices.

References

[1] J. Williams, C. Stönner, J. Wicker, N. Krauter, B. Derstroff, E. Bourtsoukidis, T. Klüpfel, and S. Kramer, "Cinema audiences reproducibly vary the chemical composition of air during films, by broadcasting scene specific emissions on breath," *Scientific Reports*, 2016.

[2] J. Wicker, N. Krauter, B. Derstorff, C. Stönner, E. Bourtsoukidis, T. Klüpfel, J. Williams, and S. Kramer, "Cinema data mining: The smell of fear," in *KDD*, 2015.

[3] http://mrbungle.zdv.uni-mainz.de/movietag/.

[4] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, 1989.

[5] K. J. Piczak, "Environmental sound classification with convolutional neural networks," in *MLSP*, 2015.

[6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F.-F. Li, "ImageNet large scale visual recognition challenge," *CoRR*, 2014.

[7] D. Tran, L. D. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," in *ICCV*, 2015.

[8] V. Hegde and R. Zadeh, "FusionNet: 3D object classification using multiple data representations," *CoRR*, 2016.

[9] F. Milletari, N. Navab, and S. Ahmadi, "V-Net: Fully convolutional neural networks for volumetric medical image segmentation," *CoRR*, 2016.

[10] M. Kohlbrenner, R. Hofmann, S. Ahmmed, and Y. Kashef, "Pre-training CNNs using convolutional autoencoders," 2017.

[11] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, 2013.

[12] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *JMLR*, 2015.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, 2014.

[14] https://www.ffmpeg.org.

[15] https://www.imagemagick.org.

[16] Itseez, https://opencv.org.

[17] B. McFee *et al.*, https://github.com/librosa/librosa/tree/0.6.0.

[18] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in Python," in *Proceedings of the 14th Python in Science Conference*, 2015.

[19] C. Harte, M. Sandler, and M. Gasser, "Detecting harmonic change in musical audio," in *Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia*, 2006.

[20] F. Chollet *et al.*, https://keras.io.

[21] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, 2012.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012.

[23] D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification," in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015.