Deep learning appliqué à la prédiction de l'émotion induite par les vidéos

Projet d'Option Informatique

Commanditaire Emmanuel Dellandrea

Professeurs encadrants Mohsen Ardabillian
Daniel Muller

Auteurs Augustin Ardon
William Blaufuks
Lucas Dormoy
Quentin Gallouédec
Timothée Haguenauer

31 mars 2020

Résumé

Ce rapport présente le travail effectué lors de notre projet d'Option Informatique. Il vise à décrire clairement les étapes que nous avons suivies pour répondre au problème de la prédiction des émotions induites par les vidéos par un modèle de *Deep Learning*, afin qu'un équipe souhaitant prendre la suite de nos recherches puissent s'approprier notre travail. Ce sujet a déjà donné lieu à un concours MediaEval en 2018 où certaines équipes de chercheurs ont participé. Dans cette étude, les émotions sont décrites par un vecteur de deux valeurs de *valence* et d'*arousal*, et de façon continue au cours du temps. L'objectif est d'entraîner un modèle de *machine learning* qui serait capable, à partir de descripteurs vidéos, de prédire l'évolution de ce signal. Trois types d'architecture ont été tentées : Un réseau fully connected, un réseau LSTM, et une combinaison de réseaux CNN sur certains descripteurs, suivies d'un réseau LSTM. Aucune de ces trois architecture ne donne de résultats satisfaisants, même s'ils sont dans le spectre des autres participants au concours MediaEval. Enfin, ce rapport donne quelques tentatives d'explication à ces résultats, ainsi que des perspectives d'amélioration, notamment au niveau des données d'entrée du modèle.

Abstract

This report presents the work carried out during our Computer Science Option project.It aims to clearly describe the steps we followed to address the problem of predicting video induced emotions using a Deep Learning model, so that a team wishing to take over our research can take ownership of our work. This topic has already given rise to a MediaEval competition in 2018 where several teams of researchers have participated. In this study, emotions are described by a two values vector of valence and arousal, and continuously over time. The objective is to train a machine learning model that would be able to predict the evolution of this signal from video descriptors. Three types of architecture have been attempted : A fully connected network, a LSTM network, and a combination of CNN networks on some descriptors, followed by a LSTM network. None of these three architectures gave satisfying results, even though they are in the spectrum of the MediaEval competition participants. Finally, this report gives some attempts to explain these results, as well as some improvment insights, especially regarding input data.

Table des matières

Table des figures

Liste des tableaux

Introduction

La prédiction des émotions induites par les vidéos est une problématique assez répandue, qui peut avoir des débouchés dans les technologies de la communication, et notamment le marketing. En 2018, le laboratoire Liris a publié un nouveau dataset constitué de séquences de films annotées en fonction des émotions induites sur le spectateur. Le but de ce projet a été de continuer les travaux déjà effectués dans le cadre de la compétition MédiaEval 2018 pour la prédiction des émotions induites par les vidéos basée sur le dataset du laboratoire Liris. Dans toute la suite du projet, nous considérerons que les émotions ressenties par un spectateur peuvent être décrites par deux valeurs : la valence, et l'excitation, respectivement *valence* et *arousal*.

Le but de ce projet est donc de construire un modèle de Deep Learning pour permettre de prédire les émotions, projetées sur ces deux axes. Nous réalisons l'apprentissage basé sur les données du laboratoire Liris, et nous nous inspirons des travaux précédemment effectués lors de la compétition par les différentes équipes en jeu.

1 Contexte du projet

1.1 Concours MEDIA-EVAL

MediaEval est une initiative de benchmarking dédiée à l'évaluation de nouveaux algorithmes pour l'accès et la récupération multimédia. Elle met l'accent sur le "multi" dans le multimédia et se concentre sur les aspects humains et sociaux des tâches multimédias. MediaEval attire des participants qui s'intéressent aux approches multimodales du multimédia impliquant, par exemple, la reconnaissance vocale, l'analyse de contenu multimédia, l'analyse musicale et audio, les informations fournies par l'utilisateur (tags, tweets), la réponse affective du spectateur, les réseaux sociaux, les coordonnées temporelles et géographiques.

Dans le cadre du projet de benchmark MediaEval, un concours est organisé chaque année sur un jeu de données créé à partir de séquences de films sous licence Creative Commons : LIRIS-ACCEDE. Cette année le jeu de données est constitué de données vidéos dites "brutes", mais aussi de *features* extraits des images et sons sur une durée cumulée d'environ 110 000 secondes, provenant d'une grande variété de films, et ce pour chaque seconde de film. L'objectif de ce concours est d'évaluer le mieux possible les émotions induites par la combinaison image-audio que constituent les données. Les émotions sont labelisée à chaque seconde dans le plan valence-*arousal*. La nouveauté de l'édition de cette année est qu'un nouveau jeu de labels a été proposé au candidats. A chaque seconde de film, on dispose d'un bit décrivant si la séquence est éffrayante ou non.

Les candidats ont donc le choix entre prédire l'émotion "générale" induite par un film, ou bien prédire si une séquence est effrayante ou non. Dans le cadre de ce projet, nous nous focaliserons sur la prédiction d'émotion générale dans le plan Valence-*arousal*.

1.2 Plan *valence-arousal*

Pour labeliser les données, le plan *valence-arousal* a été choisi. En effet, ce modèle dit "circumplex model" en anglais est très répandu pour l'évaluation d'émotions, d'expressions du visages et d'autres phénomènes que la psychologie s'est attelée à étudier au fil du temps.[8]

Le principe du plan *valence-arousal* est de dire que toute émotion peut être classifiée sur un plan circulaire avec pour axe d'abscisse la *valence*, et pour ordonnée l'*arousal*. Le centre correspond donc à une émotion de *valence* neutre avec un *arousal* moyen. On définit ainsi le plan des émotions présenté sur la figure 1.

La *valence* correspond à la "valeur" de l'émotion ; si celle-ci est positive, la *valence* sera positive. A l'inverse, si elle est négative, la *valence* sera négative

L'*arousal* correspond à la "passivité" de l'émotion. Si celle-ci est plutot passive, l'*arousal* sera faible. A l'inverse, si l'émotion est active, stimulante, l'*arousal* sera positive.

Plusieurs exemples de combinaisons de *valence*/*arousal* sont donnés dans la figure 1.

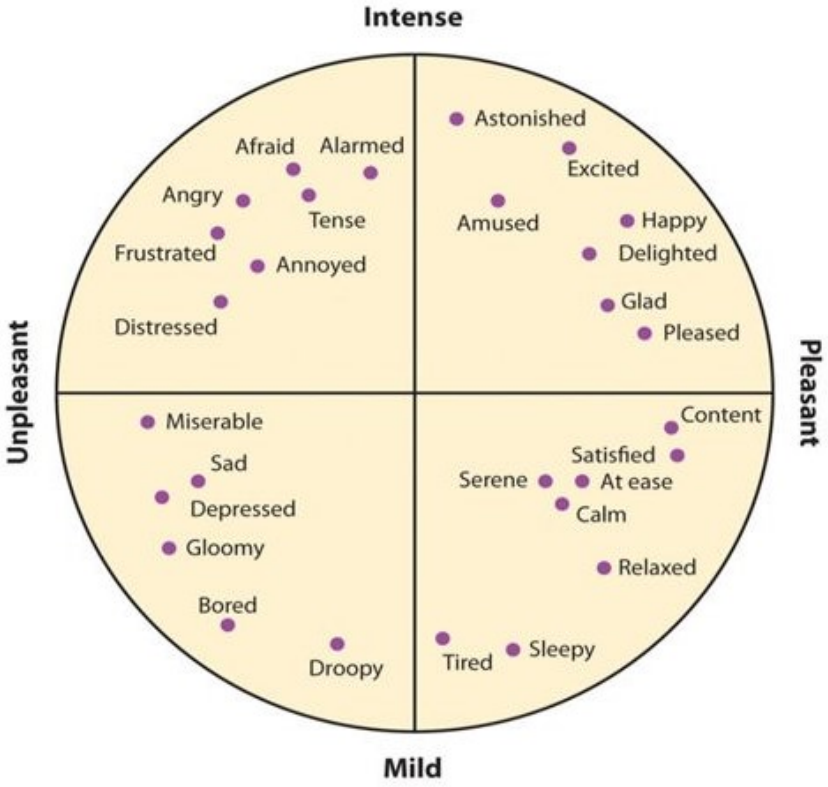


FIGURE 1 – Représentation des émotions sur un plan *valence* (abscisses) - *arousal* (ordonnées)

1.3 *Features* du jeu de données

En plus des fichiers vidéo, le LIRIS-ACCEDE dataset contient des descripteurs vidéos et audio. Dans la suite de ce rapport, ils seront aussi appelés *features*. Ce sont des *features* audios et vidéo. La description de ces *features* et de leur mode d'extraction est expliqué dans la suite.

1.3.1 *Features* audios

Les descripteurs audios sont extraits grâce à la librairie OPENSMILE. Plusieurs fonctionnelles peuvent être appliquées à ces descripteurs de bas niveau.
Pour chaque image, il y a 1 582 descripteurs évalués sur une fenêtre de 5 secondes :

- 1 428 descripteurs type LLD (21 fonctions appliquées sur 68 contours LLD (low-level descriptor))
- 152 descripteurs type pitch-based LLD (19 fonctions appliquées sur les 8 pitch-based LLD)
- 2 descripteurs : nombre de pseudo-syllabes et la taille de la séquence audio

La nomenclature ci-dessous peut être retrouvée dans les fichier `.arff` correspondant aux descripteurs audio du dataset MediaEval.

Liste des LLD calculés par OPENSMILE avec la configuration `emobase2010.conf`, utilisée pour MediaEval :

- `pcm_loudness` → Intensité normalisée élevée à la puissance 0.3
- `mfcc` → Coefficients cepstraux n°0-14 (Mel-fréquence)

— `logMelFreqBand` → Puissance logarithmique des bandes n°0-7 de Mel-fréquences (distribuées entre 0 et 8kHz)
— `lspFreq` → Les 8 "spectral line frequencies" calculées à partir de 8 coefficient LPC
— `F0finEnv` → Enveloppe du contour lissé de la fréquence fondamentale
— `voicingFinalUnclipped` → Probabilité de "voicing" du candidat final à la fréquence fondamentale
— `maxPos` et `minPos` → Positions respectives du max et du min
— `amean` → Moyenne arithmétique du contour
— `linregc1` et `linregc2` → Coefficients m et t de l'approximation linéaire du contour
— `linregerrA` et `linregerrQ` → Erreurs linéaire et quadratique entre la régression et le contour
— `stddev` → Ecart-type des valeurs du contour (moment d'ordre 2)
— `skewness` → Skewness (moment d'ordre 3)
— `kurtosis` → Kurtosis (moment d'ordre 4)
— `quartile1`, `quartile2`, `quartile3` et `quartile4` → Quartiles des valeurs du contour
— `iqr1-2`, `iqr2-3` et `iqr1-3` → Tailles des bandes inter-quartiles
— `percentile1.0` et `percentile99.0` → Pourcentiles à 1% et 99%
— `pctlrange0-1` → Taille de la bande entre les deux paramètres précédents
— `upleveltime75` et `upleveltime90` → Durée pendant laquelle le signal est au dessus de (75%*amplitude + min) et (90%*amplitude + min) respectivement
— `F0final` → Contour lissé de la fréquece fondamentale
— `jitterLocal` → Guigue (jitter)
— `jitterDDP` → Guigue de la guigue (jitter differentiel)
— `shimmerLocal` → Shimmer (perturbations à court terme de l'amplitude du signal)

Pour augmenter ces données, les deltas de certains de ces descripteurs sont ajoutés, ainsi que certaines de leurs moyennes glissantes sur plusieurs tailles de plage. Les fonctionnelles appliquées à ces descripteurs sont les suivantes :
— Extrema : valeurs et positions
— Moyennes (arithmétique, quadratique, géométrique)
— Moments (ecart-type, variance, kurtosis, skewness (asymétrie))
— Pourcentiles et valeurs aux pourcentiles
— Regressions (approximation linéaire et quadratique, erreur de regression)
— Centroïde
— Pics d'amplitude
— Segments
— Valeurs d'échantillon
— Temps/durées
— Début/Décalage
— Transformée en cosinus discrète
— Passages par le zéro
— Linear Predictive Coding (LPC) coefficients et gain

Le détail de ces descripteurs et de leur extraction peuvent être trouvés dans le OpenSmile book [4].

1.3.2 *Features* vidéos

Pour chaque seconde du film, une image est extraite. Sur cette image, les descripteurs suivant sont calculés :

— *Auto Color Correlogram* (`acc`)
— *Color and Edge Directivity Descriptor* (`cedd`)
— *Color Layout* (`cl`)
— *Edge Histogram* (`eh`)
— *Fuzzy Color and Texture Histogram* (`fcth`)
— *Gabor* (`gabor`)
— *Joint descriptor* (`jcd`)
— *Scalable Color* (`sc`)
— *Tamura* (`tamura`)
— *Local Binary Patterns* (`lbp`)
— *VGG16 fc6 layer output* (`fc6`)

Dans la suite, on explique ce que sont ces descripteurs et comment ils sont extraits.

Auto Color Correlogram L'*Auto Color Correlogram* est une matrice de taille (m, n). La coordonnée i correspond à une couleur, tandis que la coordonnée j correspond à une distance en pixels. La valeur du coefficient (i, j) donne la probabilité d'occurence d'un pixel de couleur i à partir d'un pixel de même couleur, à une distance d. [12]

Les implémentations de l'*Auto Color Correlogram* prennent systématiquement 64 couleurs. Dans notre cas, 4 valeurs de distances ont été choisies. Ainsi, le résultat est une matrice de taille $(64, 4)$.

Color and Edge Directivity Descriptor Le *Color and Edge Directivity Descriptor* est un histogramme constitué de 6 régions [3]. Chaque région correspond à une texture. Les textures sont déduites par l'application de filtres. Ces 6 régions sont correspondent aux caractéristiques suivantes : pas de bord, bord non directionnel, bord horizontal, arête verticale, diagonale de 45 degrés, diagonale de 135 degrés. Pour chacune de ces textures, on distingue 24 régions individuelles. Chacune de ces sous-régions émanent d'une unité de couleur. La concaténation de toutes ces valeurs donne un vecteur de taille 144 valeurs.

Color Layout Le *Color Layout* est conçu pour saisir la distribution spatiale des couleurs dans une image [11]. L'extraction des caractéristiques commence par la sélection représentative des couleurs sur la grille, puis la transformation en cosinus discret avec quantification.

Edge Histogram L'*Edge Histogram* représente la fréquence relative d'apparition de 5 types de bords dans chaque zone locale appelée bloc d'image [13]. Le bloc d'image est défini en divisant l'espace image uniformément en 16 blocs non chevauchants. La distribution des bords pour chaque bloc d'image est générée. Ils sont de 5 types : vertical, horizontal, diagonal à 45 degrés, diagonal à 135 degrés et bords non directionnels. L'histogramme représente la distribution relative des 5 types de bords dans le bloc image correspondant.

Fuzzy Color and Texture Histogram Le *Fuzzy Color and Texture Histogram* est un descripteur bas niveau utilisé pour la récupération d'images, en limitant l'impact des distorsions tels que les déformations [5]. Il est constitué de 8 régions, correspondant à 8 types de texture.

Chaque région est constituée de 24 régions individuelles, correspondant chacune à une couleur. Au total, la sortie qui en résulte comprend 192 valeurs.

Gabor Le filtre de *Gabor* est un filtre linéaire utilisé pour l'analyse de texture [7]. Pour une fréquence et une direction donnée, il détecte la correspondance autour du point considéré dans l'image.

Joint descriptor Le *Joint descriptor* combine, dans un seul histogramme, des informations de couleur et de texture [14]. Il est composé des descripteurs *Fuzzy Color and Texture Histogram* et *Color and Edge Directivity Descriptor*. C'est un descripteur composite compact (CCD) pour la récupération d'images basée sur le contenu.

Scalable Color Le descripteur *Scalable Color* se base sur l'histogramme *Structuring Window* (SW) et l'exprime dans le domaine des fréquences. L'accumulation traditionnelle d'histogrammes est appliquée sur l'image. Les résultats sont transformés dans le domaine des fréquences en utilisant le filtre de Haar [2].

Tamura Les caractéristiques de Tamura [10] correspondent à des élément de texture comme le contraste, la directionnalité et la rugosité. Ce choix est est basée sur des études psychophysiques des éléments caractéristiques qui sont perçus dans les textures par les humains.

Local Binary Patterns Les *Local Binary Patterns* sont un type de descripteur visuel utilisé pour la classification dans le domaine de vision par ordinateur [1]. En fixant un seuil pour le voisinage de chaque pixel, il permet d'étiqueter les pixels d'une image de considérer le résultat comme un nombre binaire. Cet opérateur est robuste aux changements d'échelle de gris résultant, par exemple, de variations d'illumination.

VGG16 fc6 layer output Le VGG16 est un modèle de réseau neuronal convolutif [9] utilisé pour la classification de contenu. Le modèle atteint une précision de 92,7% de précision sur le dataset d'ImageNet. Les trois dernières couches sont des couches *fully-connected* et sont appelées *fc6*, *fc7*, *fc8*. Nous utilisons ici la première de ces couches.

1.4 Etat de l'art MEDIA-EVAL

1.4.1 Modes d'évaluation

Le concours MediaEval souhaitait évaluer la performance des solutions proposées selon deux crièteres :

Erreur quadratique moyenne Soit un vecteur de n prédictions généré à partir d'un échantillon de n points de données sur toutes les variables. On note Y le vecteur des valeurs réelles et $\hat{Y}$ le vecteur des valeurs prédites. L'erreur quadratique moyenne *MSE*, est la norme euclidienne calculant la distance moyenne entre la valeur prédite et la valeur réelle. Elle se calcule par la

relation suivante :

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{1}$$

Ainsi, l'objectif est d'obtenir une MSE la plus faible possible pour avoir un algorithme performant.

Coefficient de corrélation de Pearson Le coefficient de corrélation de Pearson r mesure la corrélation linéaire entre les valeurs prédites et les valeurs réelles. Ce coefficient est compris entre -1 et 1, et l'objectif est de se rapprocher le plus possible de 1.

$$r = \frac{\text{cov}(Y, \hat{Y})}{\sigma_Y \sigma_{\hat{Y}}} \tag{2}$$

Ces deux critères ont l'avantage d'être complémentaires (l'un renseigne une valeur de performance, et l'autre une valeur de dispersion), et d'avoir été utilisés par tous les groupes ayant travaillé sur le projet. Il sera ainsi facile de comparer les résultats obtenus aux valeurs déjà existantes.

1.4.2 Etat de l'art des participants

Durant les quatres dernières éditions du concours MediaEval, les équipes ont utilisé des techniques variées de Machine Learning pour prédire au mieux la *valence* et l' *arousal* en continue pour une vidéo donnée. Plusieurs modèles ont été utilisés :

- Modélisation statique : *K-nearest neighbors* (KNN), *Support Vector Regression*, *Neural Networks*.

- Modélisation temporelle : *Long Short-Term Memory* (LSTM, en simple ou en bidirectionnel), *Gated Recurrent Unit* (GRU).

Par ailleurs, les participants ont choisis de prendre en entrée de leurs algorithmes, soit les features visuels et/ou audio, soit les vidéos elles-mêmes, soit la combinaison des deux. Cela a permis d'obtenir des approches complètement différentes dans la compréhension des émotions ressenties par le spectateur.

Les résultats les plus pertinents de ces groupes sont présentés dans la table 1. Les données en gras représentent les meilleures valeurs en terme de MSE et Pearson (respectivement pour la *valence* puis pour l'*arousal*) obtenues jusqu'à présent.

	Valence		Arousal	
	MSE	r	MSE	r
Yun Yi et al.	0.09142	0.27518	0.14634	0.11571
E. Batziou et al.	0.117	0.098	0.138	nan
Ye Ma et al.	0.0924	**0.3048**	0.1399	0.0761
K. C. Quan et al.	0.11504	0.14565	0.17055	0.07525
J. J. Sun et al.	**0.0837**	0.1786	**0.1334**	**0.3358**
T. H. Ko et al.	0.1089	0.0872	0.1608	0.0487

TABLE 1 – Résultats de la dernière édition du concours MediaEval

2 Choix architecturaux et expérimentations

3 architectures différentes ont été envisagées. En partant de la plus simple, cette partie explique notre démarche pour choisir ces architectures et nous présentons les résultats obtenus dans chaque cas. La fonction de perte sera systématiquement *Mean-Square Error* (MSE). Un modèle prédisant systématiquement la moyenne a une MSE sur le dataset de test de 0.135358.

2.1 Fully connected

La première architecture envisagée est celle d'un réseau *fully connected*. A chaque seconde, une *frame* est extraite, ainsi qu'une bande sonore de 5 secondes centrée sur cet instant. Des *features* sont extraits de cette *frame* et de cette bande sonore. Pour un instant donnée, 6950 *features* sont ainsi générées. Les *features* et leur construction sont décrites dans la partie 1.3. A défaut de faire une prédiction précise, cette architecture simple nous donnera une base pour comparer nos prochains résultats.

La figure 2 montre l'architecture envisagée.

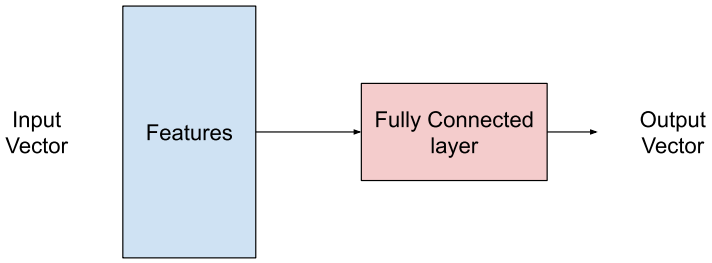
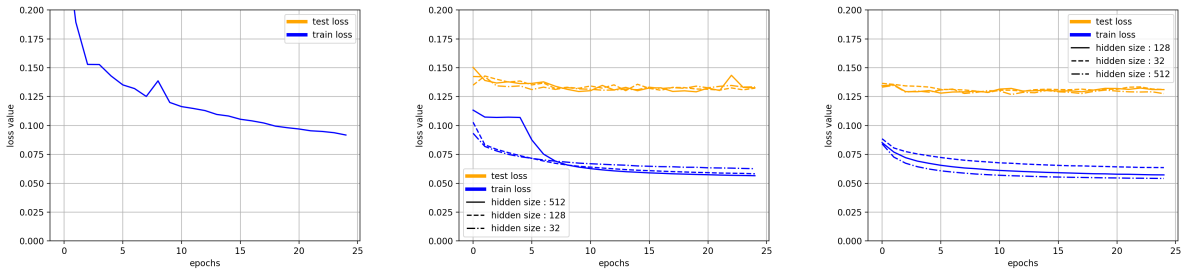


FIGURE 2 – Schéma de l'architecture fully connected

Les essais ont été réalisés avec 1, 2 et 3 couches cachées. Plusieurs tailles de couches cachées ont également été envisagées. La figure 3 présente les résultats.



(a) 0 couche cachée. Nombre de paramètres : 13 902

(b) 1 couche cachée. Nombre de paramètres : 222 498, 889 986 et 3 559 938

(c) 2 couches cachées, Nombre de paramètres : 223 554, 906 498 et 3 822 594

FIGURE 3 – Courbes d'entraînement pour un réseau *fully connected* avec 0, 1 et 2 couches cachées. L'ensemble des *features* est pris en entrée. Calcul de la *loss* : MSE. Entraînement : 25 *epoch*. *Gradient clipping* : 1. *Optimizer* : RMSprop algorithm (*learning rate* : 10^{-4} , *momentum factor* : 0, *smoothing constant* : 0.99, *L2 penalty* : 10^{-5})

Il est intéressant de remarquer que les modèles à 1 et 2 couches cachées, la *test loss* converge vers une valeur finale autour de 0.13, tandis que *train loss* se rapproche de 0.05. Malgré le terme

de régularisation et le *dropout*, le modèle semble souffrir d'*overfitting*.

Pour limiter ce le phénomène d'*overfitting*, nous voulons identifier les *features* qui influent le plus sur la sortie. Éliminer les autres *features* nous permettrait de limiter le nombre d'entrées, et ainsi réduire le phénomène d'*overfitting*.

L'évolution des *test loss* obtenues pour l'entraînement de réseaux fully connected qui prennent en entrée une unique *feature* sont présentés figure 4.

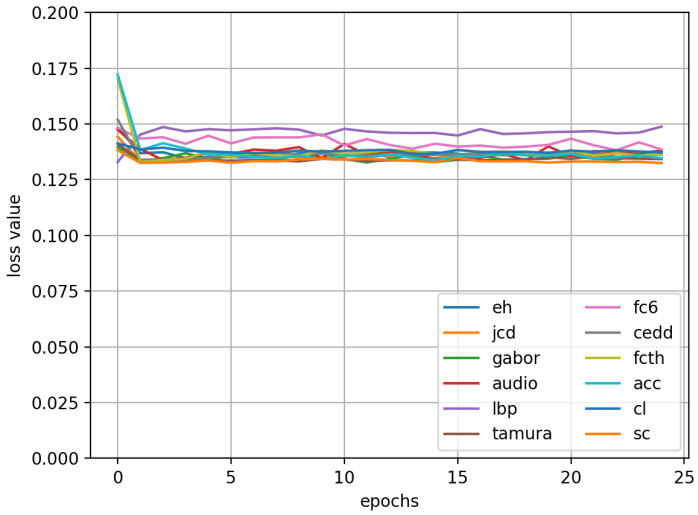


FIGURE 4 – *test loss* obtenue sur l'entraînement d'un réseau *fully connected* à 1 seule feature. Réseau *fully connected* avec 1 couche cachée. Une unique *feature* est prise en entrée. Calcul de la *loss* : MSE. Entraînement : 25 *epoch*. *Gradient clipping* : 1. *Optimizer* : RMSprop algorithm (*learning rate* : 10^{-4} , *momentum factor* : 0, *smoothing constant* : 0.99, *L2 penalty* : 10^{-5})

Aucune *feature* ne semble donner de meilleurs résultats que les autres. Il est fort probable que le traitement des *features* à un instant donné ne suffise pas à déduire précisément l'émotion induite par une séquence d'images, ou bien que la grande quantité de ces *features* et leur redondance noient le signal recherché dans du bruit.

La manière dont s'enchaînent les images et les sons ont très certainement une influence bien plus importante sur l'émotion induite chez le spectateur. La prochaine section décrit comment nous intégrerons au modèle la dimension temporelle, en utilisant un réseau récurent.

2.2 Modèle récurrent LSTM

Afin de prendre en compte l'enchainement des *frames*, nous avons donc mis en place un réseau récurrent. Nous avons choisi l'architecture LSTM, qui est connue pour donner de bons résultats dans des applications de prédiction basées sur des séquences temporelles.

Nous avons donc testé un modèle LSTM comme présenté dans la figure 5, encore une fois en faisant varier le nombre et la taille des couches cachées, pour déterminer quelle taille de réseau serait la mieux adaptée à notre problème. Les résultats de ces essais sont présentés figure 6.

On voit sur ces courbes que la *test loss* atteint très vite un plancher, alors que la *train loss* continue de diminuer. Donc, on arrive rapidement dans une situation d'*overfitting*. Ainsi, on essaie de déterminer quels *features* influent le plus sur la sortie, afin de les isoler et donc limiter ce phénomène. L'évolution des *test loss* obtenues pour l'entraînement de réseaux LSTM qui prennent en entrée une unique *feature* sont présentés figure 7.

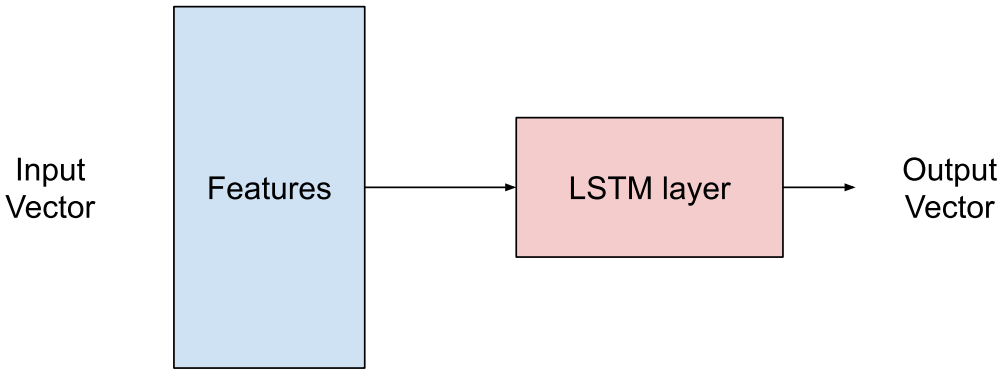
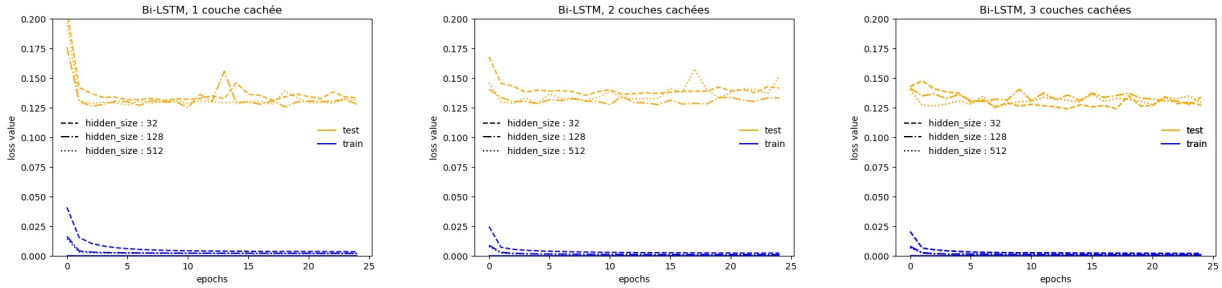


FIGURE 5 – Schéma de l'architecture LSTM



(a) 1 couche cachée (b) 2 couches cachées (c) 3 couches cachées

FIGURE 6 – Courbes d'entraînement pour un réseau Bi-LSTM avec 1, 2 et 3 couches cachées. L'ensemble des *features* est pris en entrée. Calcul de la *loss* : MSE. Entraînement : 25 *epochs*. *Gradient clipping* : 1. *Optimizer* : RMSprop algorithm (*learning rate* : 10^{-4} , *momentum factor* : 0, *smoothing constant* : 0.99, *L2 penalty* : 10^{-5}). Dropout : 0.3.

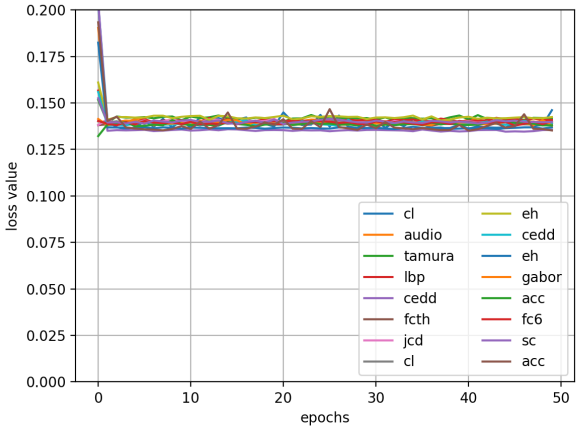


FIGURE 7 – *test loss* obtenue sur l'entraînement d'un réseau BiLSTM à 1 seule *feature* (voir légende). 1 couche cachée de taille 32. Longueur des séquence : 64. Calcul de la *loss* : MSE. Entraînement : 50 *epoch*. *Gradient clipping* : 1. *Optimizer* : RMSprop algorithm (*learning rate* : 10^{-3} , *momentum factor* : 0, *smoothing constant* : 0.99, *L2 penalty* : 10^{-4}). Dropout : 0.4.

Cette figure nous permet d'observer qu'aucun *feature* ne se démarque par rapport aux autres dans son influence sur la prédiction. Nous avons donc cherché à travailler sur ces *features* afin de leur donner plus de sens.

2.3 CNN puis LSTM

Les données en entrée sont hétérogènes. Elles correspondent à des descripteurs de taille et de construction différentes. Dans certains cas, le descripteur calculé correspond à une représentation spatiale de l'image. Par exemple, l'*Edge Histogram* décrit la prévalence de différents type de contour sur 16 zones de l'image. Ce descripteur peut donc être vu comme une image 4 par 4 à 5 *channels* (1 pour chaque type de contour). En ajoutant un premier étage de convolutions 2D sur cette image avant la couche récurrente, on peut supposer que le résultat obtenu pour la phase d'entraînement sera meilleur. La figure 8 présente l'architecture CNN utilisée en amont de la couche LSTM.

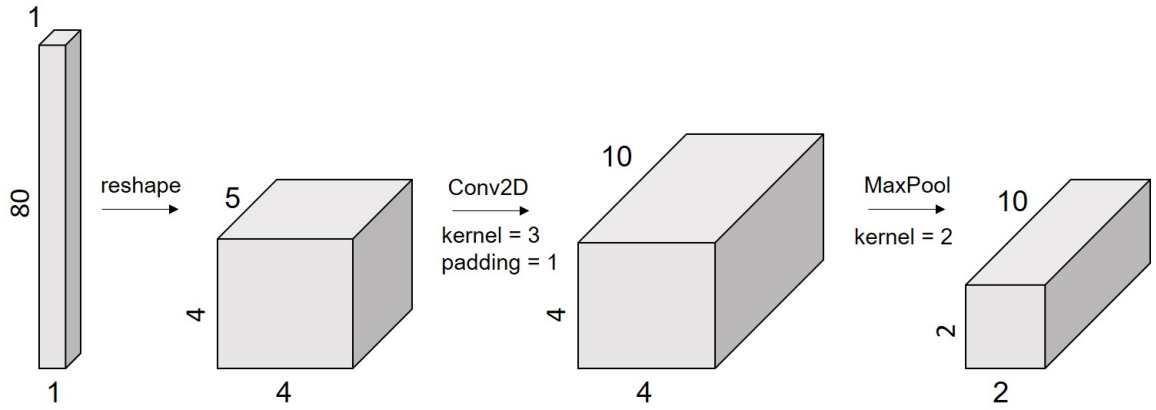


FIGURE 8 – Réseau EH

Dans le même raisonnement, d'autres features décrivent une information continue dans une dimension. Le *Color and Edge Directivity Descriptor* decrit pour un *continuum* de 24 couleurs, la prévalence de 6 types de contour. En le considérant les type de contour comme des *channels*, on obtient un tenseur de dimension (24, 1, 6). Pour les mêmes raison que précédemment, on envisage donc un premier étage de convolution 1D, en amont de la couche LSTM. La figure 9 présente l'architecture CNN utilisée pour le *Color and Edge Directivity Descriptor*.

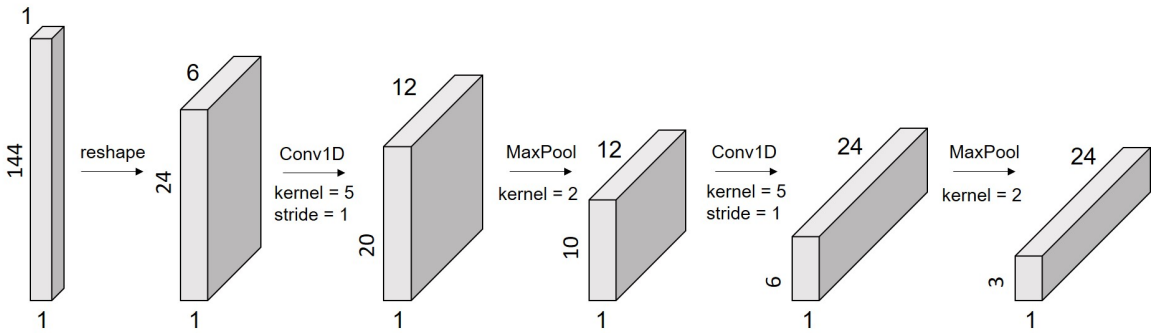


FIGURE 9 – Réseau CEDD

Les descripteurs *Auto Color Correlogram*, *Fuzzy Color and Texture Histogram* et *Joint descriptor* sont construits d'une façon très similaire au *Color and Edge Directivity Descriptor*. Les figures 10, 11 et 12 présentent respectivement les architectures CNN que nous avons envisagées.

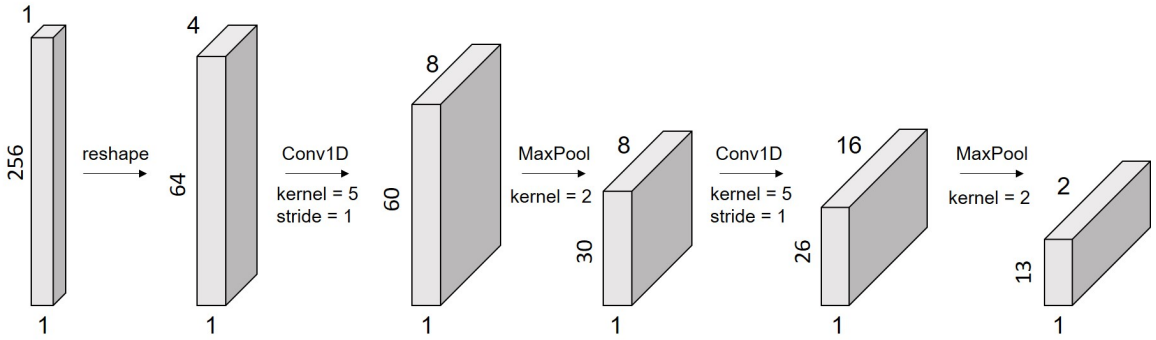


FIGURE 10 – Réseau ACC

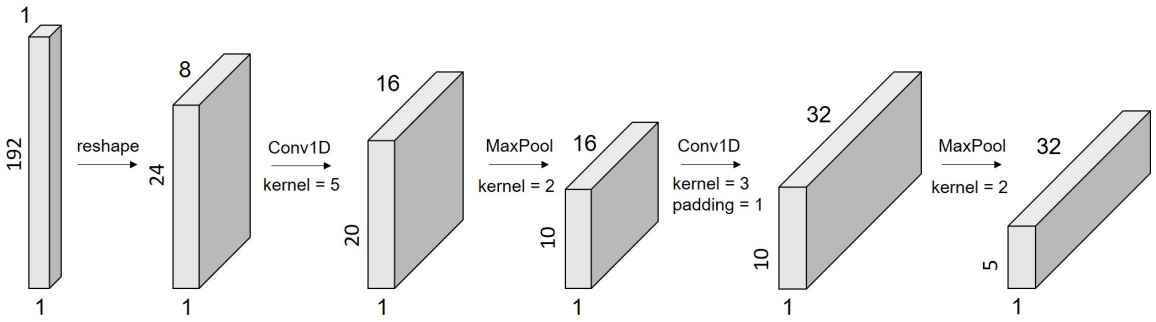


FIGURE 11 – Réseau FCTH

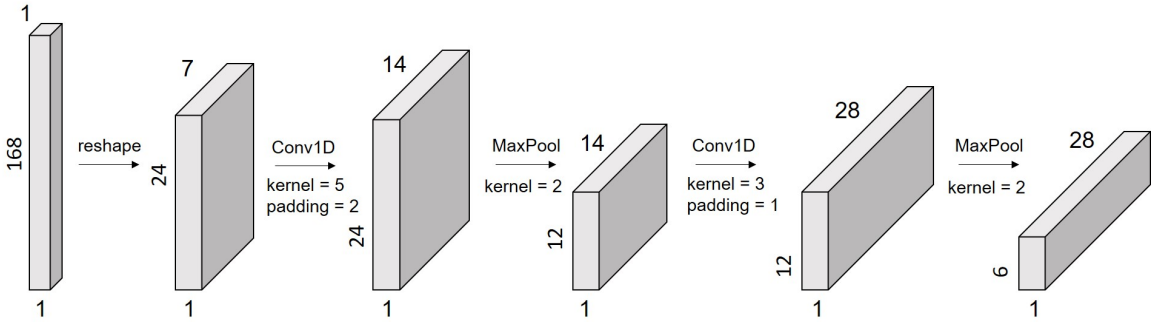


FIGURE 12 – Réseau JDD

Pour les autres *features*, soit leur construction ne présente pas de continuum dans une dimension (par exemple *VGG16 fc6 layer output*), soit nous n'avons pas su trouver assez d'informations à propos de leur construction.

En aval des étages de CNN, une couche de LSTM et deux couches *fully connected* sont connectées. Entre chaque couche *fully connected* et dans la couche LSTM, il y a une couche de *dropout* pour limiter l'*overffiting*. Le schéma du réseau obtenu est présenté figure 13.

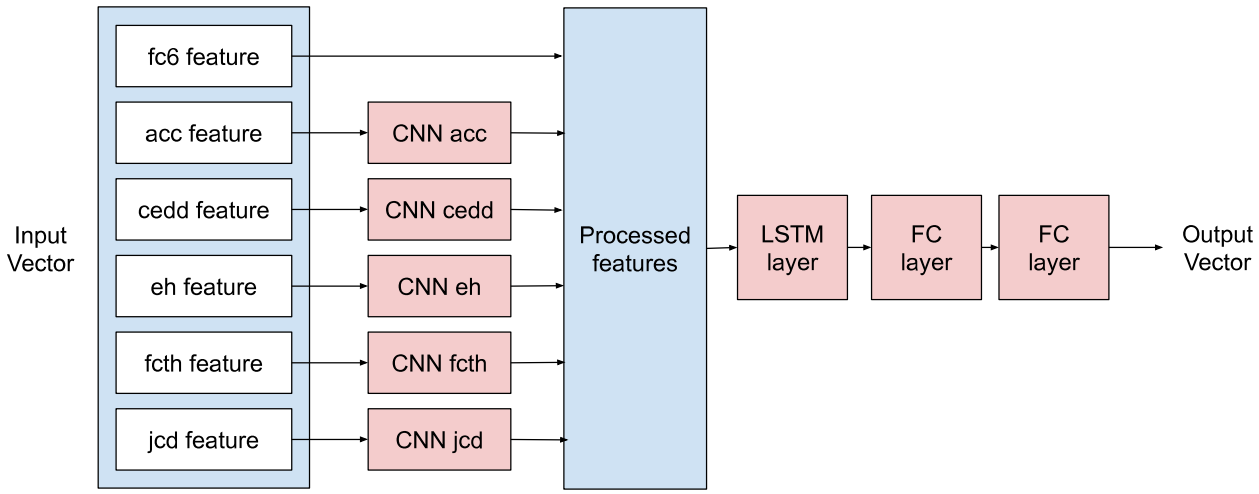


FIGURE 13 – Réseau avec une première couche de CNN pour certains descripteurs puis une couche LSTM

Le résultat de l'entraînement est donné figure 14.

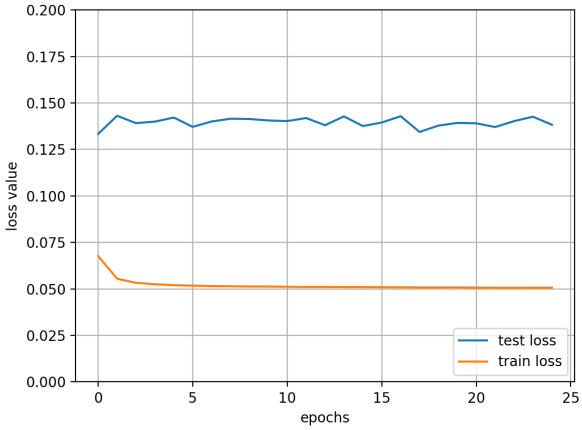


FIGURE 14 – Courbes d'entraînement obtenues sur un réseau CNN-BiLSTM. 2 couches cachées de taille 128. Longueur des séquences : 16. Calcul de la *loss* : MSE. Entraînement : 25 *epoch*. *Gradient clipping* : 1. *Optimizer* : RMSprop algorithm (*learning rate* : 10^{-3} , *momentum factor* : 0, *smoothing constant* : 0.99, *L2 penalty* : 10^{-5}). Dropout : 0.3. Nombre de paramètre du modèle 5 426 446.

Il n'y a pas d'amélioration significative de la précision du modèle. La *test loss* converge vers une valeur finale autour de 0.13.

Sur la table 2, nous comparons nos résultats à ceux trouvés dans la littératures, introduits dans la partie 1.4.2. On remarque que nos résultats restent malgré tout assez proches de ceux obtenus par les participants de la dernières édition du concours MediaEval. Malgré les diverses méthodes employées, on remarque que prédire l'émotion induite est une tâche complexe, et qu'il est difficile de donner une prédiction précise.

	Valence		Arousal	
	MSE	r	MSE	r
Modèle constant moyen	0.1334	nan	0.1373	nan
Yun Yi et al.	0.09142	0.27518	0.14634	0.11571
E. Batziou et al.	0.117	0.098	0.138	Nan
Ye Ma et al.	0.0924	**0.3048**	0.1399	0.0761
K. C. Quan et al.	0.11504	0.14565	0.17055	0.07525
J. J. Sun et al.	**0.0837**	0.1786	**0.1334**	**0.3358**
T. H. Ko et al.	0.1089	0.0872	0.1608	0.0487
ECL projet 6	**0.1173**	**0.2789**	**0.1614**	**-0.0304**

TABLE 2 – Résultats de la dernière édition du concours MediaEval, comparées avec nos résultats.

3 Description du code

L'ensemble du code du projet est hébergé sur un répertoire public Github à l'adresse suivante : `https://github.com/KongHag/emotion_project`

3.1 Utilisation du code

L'utilisation du code est explicitée dans les différents fichiers `README.md` du repository GitHub ainsi que dans l'annexe A.

3.2 Structure du répertoire

Le répertoire est structuré en fichiers python distincts où chacun est responsable d'une tâche. La définition d'un modèle et l'entraînement de ce dernier pour obtenir les résultats présenté précédemment est réalisé grâce à l'exécution du fichier **emotion.py**. Ce fichier appelle successivement les classes et fonction définies dans les autres fichiers. Ces fichiers sont organisés de la façon suivante :

- **dataset.py** - Définit une nouvelle classe dataset permettant de charger les données d'entraînement de et de test.
- **emotion.py** - Définit l'exécutable appelant les phases de définitions du dataset, du modèle puis la phase d'entraînement. Définit de même un *argparser* permettant de passer des arguments du modèle à la phase d'exécution.
- **log.py** - Définit un objet *logger* permettant de tracer l'exécution du programme dans un fichier externe.
- **model.py** - Définit le modèle utilisé.
- **read_data.py** - Définit des fonctions permettant de faire un pré-traitement des données.
- **schedule_train.py** - Définit une autre source exécution semblable à celle d'**emotion.py** en définissant les paramètres globaux du modèle sous forme de *json* plutôt que comme arguments d'un *argparser*.
- **training.py** - Définit la phase d'entraînement du modèle.

A l'issue d'un entraînement, les résultats de loss sur les données de test et d'apprentissage sont disponible dans le dossier *results*. Il sont stockés sous forme d'un fichier *json* comportant les paramètres du réseau utilisé et les résultats des calculs de loss au long de l'apprentissage. Un exemple d'un tel fichier peut être trouvé en annexe B

Un utilitaire d'affichage de graphiques est disponible dans ce même dossier. Il s'agit du fichier python exécutable **show.py**. Il permet de générer les graphiques de tous les fichiers *json* disponibles. Les graphiques sont alors générés dans le dossier *results/graphs*.

3.3 Surcharge des classes PyTorch

L'ensemble du projet a été réalisé à l'aide du framework de deep learning **Pytorch**. De cette manière, nous avons défini des fonctions pour l'entraînement et surchargé des classes prédéfini dans Pytorch.

3.3.1 Le dataset

La classe Dataset par défaut de Pytorch a été surchargée de manière à charger les données du concours MédiaEval. La classe prend désormais en argument :

— **root** (str, optional) : default './data'. Chemin vers les données au format **pickle**
— **train** (bool, optional) : default True. il s'agit de l'apprentissage si True, le test sinon
— **seq_len** (int, seq_len) : default 100. Nombre d'images par échantillon
— **shuffle** (bool, optional) : default False. Mélanger les données ou non
— **fragment** (float, optional) : default 1. De 0 à 1, le pourcentage des données utilisées
— **overlapping** (bool, optional) : Autoriser les données à se superposer

La classe `MediaEval18(Dataset)` a été créée de manière à définir les données. Le but est de pouvoir définir un **Dataloader** de Pytroch pour réaliser un apprentissage.

3.3.2 Le modèle

La classe Module de Pytorch a été surchargée de manière à définir le modèle que nous avons présenté précédemment. La classe prend en argument :

— **input_size** (int) – Taille de la couche d'entrée (Deprecated)
— **hidden_size** (int) – Taille de la couche cachée
— **num_layers** (int) – Nombre de couche pour le LSTM
— **output_size** (int) – Taille du vecteur de sortie
— **dropout** (float) – Probabilité de dropout (entre 0 et 1)
— **bidirectional** (bool) – Utiliser ou non un réseau LSTM bidirectionnel

Le réseau utilisé est alors celui présenté dans les parties précédentes. Il peut être utilisé pour réaliser un entraînement.

3.3.3 L'entraînement

L'entraînement du réseau a été défini à l'aide d'une fonction python. Elle opère de la manière suivante. Pour chaque epoch et pour chaque batch dans le dataset d'entraînement :

— Envoyer les tenseurs sur le GPU s'il est disponible
— Calcul de la sortie du réseau
— Calcul de la loss en fonction des paramètres de score
— Rétro-propager l'erreur à travers le réseau pour le cas de l'entraînement

Cette fonction prend en argument :

— **model** (`torch.nn.Module`) – modèle utilisé

— **trainloader** (`torch.utils.data.DataLoader`) – dataloader de pytorch contenant les données d'entraînement
— **testloader** (`torch.utils.data.DataLoader`) – dataloader de pytorch contenant les données de test
— **criterion** (`torch.nn.modules.loss._Loss`) – critère de score utilisé pour calculer la loss
— **optimizer** (`torch.optim.optimizer.Optimizer`) – optimiseur du réseau
— **device** (`str`) – device utilisé (gpu / cpu)
— **grad_clip** (`float`) – valeur maximale absolue du gradient
— **nb_epoch** (`int`) – Nombre d'epochs d'entraînement

A l'aide de cette fonction, l'entraînement du réseau défini précédemment, avec les données définies précédemment est réalisé, à partir des paramètres d'optimiseur et de fonction score données en argument de l'exécution de **emotion.py**.

Conclusion

Trois approches différentes ont été tentées pour parvenir à des résultats. Aucune de ces trois approches n'a donné des résultats significativement meilleurs que ceux que l'on obtient en prédisant systématiquement l'émotion moyenne sur le dataset. Nous retenons malgré tout le modèle CNN + LSTM comme étant le plus prometteur, qui serait le plus intéressant à creuser pour la suite de ce travail. Toutefois, malgré de nombreuses tentatives d'affinage des hyper-paramètres, aucun modèle n'a vraiment donné satisfaction, en stagnant toujours à un score MSE de 0.13.

Certaines options restent à envisager pour améliorer ce score au maximum, comme tenter une architecture GRU ou une architecture récurrente avec attention, ou encore continuer le travail d'extraction des descripteurs, et leur exploitation avec des réseaux CNN. Il serait également intéressant de travailler directement sur les données brutes, plutôt que sur les descripteurs. Puisque notre score est proche des scores obtenus par les participants au concours MediaEval, il est probable que les descripteurs ne permettent pas d'extraire l'émotion induite. C'est peut-être l'étage d'extraction des descripteurs qui détruit le signal "émotion".

A Exécution de `emotion.py`

Pour lancer un entraînement, il faut éxécuter le fichier `emotion.py` de la manière suivante :

```
usage: python emotion.py [-h] [--model {FC,LSTM,CNN_LSTM}] [--seq-len SEQ_LEN]
                         [--num-hidden NUM_HIDDEN] [--hidden-size HIDDEN_SIZE]
                         [--lr LR] [--batch-size BATCH_SIZE] [--grad-clip GRAD_CLIP]
                         [--nb-epoch NB_EPOCH] [-O {Adam,RMSprop,SGD}] [-B BIDIRECT]
                         [--weight-decay WEIGHT_DECAY] [-D DROPOUT]
                         [--logger-level LOGGER_LEVEL] [--fragment FRAGMENT]
                         [--features {acc,cedd,cl,eh,fcth,gabor,jcd,sc,
                         tamura,lbp,fc6,visual,audio,all}
                         [{acc,cedd,cl,eh,fcth,gabor,jcd,sc,
                         tamura,lbp,fc6,visual,audio,all} ...]]
                         [--no-overlapping]

Train Neural Network for emotion predictions

optional arguments:
  -h, --help            show this help message and exit
  --model {FC,LSTM,CNN_LSTM}
                        Type of model. Default LSTM.
  --seq-len SEQ_LEN     Length of a sequence
  --num-hidden NUM_HIDDEN
                        Number of hidden layers in NN
  --hidden-size HIDDEN_SIZE
                        Dimension of hidden layer
  --lr LR               Learning rate
  --batch-size BATCH_SIZE
                        Size of a batch
  --grad-clip GRAD_CLIP
                        Gradient clipped between [- grad-clip, grad-clip]
  --nb-epoch NB_EPOCH   Number of epoch
  -O {Adam,RMSprop,SGD}, --optimizer {Adam,RMSprop,SGD}
                        Type of optimizer
  -B BIDIRECT, --bidirect BIDIRECT
                        Whether to use bidirectional
  --weight-decay WEIGHT_DECAY
                        L2 regularization coefficient
  -D DROPOUT, --dropout DROPOUT
                        Dropout probability between [0, 1]
  --logger-level LOGGER_LEVEL
                        Logger level: from 10 (debug) to 50 (critical)
  --fragment FRAGMENT   The percentage of the dataset used. From 0 to 1
  --features {acc,cedd,cl,eh,fcth,gabor,jcd,sc,tamura,lbp,fc6,visual,audio,all}
             [{acc,cedd,cl,eh,fcth,gabor,jcd,sc,tamura,lbp,fc6,visual,audio,all} ...]
                        Features used
  --no-overlapping      Forbid overlapping between sequences in dataset
```

B Format des résultats d'entraînement

Les résultats des entraînement sont envoyés dans le dossier `results/` au format `.json`. Voici un exemple.


```
{
    "seq_len":16,
    "num_hidden":2,
    "hidden_size":128,
    "lr":0.0001,
    "batch_size":64,
    "grad_clip":1,
    "nb_epoch":25,
    "optimizer":"RMSprop",
    "crit":"MSE",
    "weight_decay":1e-05,
    "bidirect":true,
    "dropout":0.3,
    "logger_level":20,
    "fragment":1,
    "features":"all",
    "overlapping":true,
    "model":"CNN_LSTM",
    "MSE_valence":0.11735366284847,
    "MSE_arousal":0.16144749522209,
    "r_valence":0.27891695,
    "r_arousal":-0.03035881,
    "train_losses":[
        0.06766780524999462,
        0.0554808939036247,
        0.05328191127279278,
        0.0525017321821344,
        0.052024716430444845,
        0.05175124549408084,
        0.051532746202657735,
        0.05141200439425443,
        0.051303606378507526,
        0.0512613932793953,
        0.051137696039140895,
        0.05102777184311594,
        0.05102428128238251,
        0.05096026515720712,
        0.050949425913030734,
        0.05089634834938728,
        0.05086747980659151,
        0.05077556314484011,
        0.050787132382671935,
        0.05080624341908912,
        0.05070442516872945,
        0.050670994936805984,
        0.05066729605142097,
        0.05068221000417342,
        0.0506814202272825
    ],
    "test_losses":[
        0.13335648107528686,
        0.14309982411563396,
        0.13913739386200905,
        0.13994607286155225,
        0.14214669683575631,
        0.13713499373197555,
        0.14002976806461812,
        0.14150513346493243,
        0.14136116518080236,
        0.14057547426223754,
        0.14023509326577185,
        0.14188315868377685,
        0.13801889573037623,
        0.14277161745727063,
        0.13757741416990757,
        0.13946284282207488,
        0.14287161906063556,
        0.13440595211088657,
        0.13782079808413983,
        0.13925490568578244,
        0.13903685569763183,
        0.13703640818595886,
        0.1402274762094021,
        0.14259698790311814,
        0.13827179361879827
    ]
}
```

Références

[1] Timo Ahonen, Abdenour Hadid et Matti Pietikainen : Face description with local binary patterns : Application to face recognition. *IEEE transactions on pattern analysis and machine intelligence*, 28(12):2037–2041, 2006.

[2] Jing-Ying Chang, Chung-Jr Lian et Liang-Gee Chen : Architecture and analysis of color structure and scalable color descriptor for real-time video indexing and retrieval. *In IEEE International Symposium on Consumer Electronics, 2004*, pages 365–369. IEEE, 2004.

[3] Savvas A Chatzichristofis et Yiannis S Boutalis : Cedd : color and edge directivity descriptor : a compact descriptor for image indexing and retrieval. *In International Conference on Computer Vision Systems*, pages 312–322. Springer, 2008.

[4] Martin Wöllmer Björn Schuller Florian Eyben, Felix Weninger : *Latest OpenSmile Book*. OpenSmile, 2018.

[5] Ju Han et Kai-Kuang Ma : Fuzzy color histogram and its use in color image retrieval. *IEEE transactions on image processing*, 11(8):944–952, 2002.

[6] Song Han, Huizi Mao et William J Dally : Deep compression : Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv :1510.00149*, 2015.

[7] Rajiv Mehrotra, Kameswara Rao Namuduri et Nagarajan Ranganathan : Gabor filter-based edge detection. *Pattern recognition*, 25(12):1479–1494, 1992.

[8] James A. Russell : A circumplex model of affect. *Journal of Personality and Social Psychology*, 39(6):1161—-1178, 1980.

[9] Karen Simonyan et Andrew Zisserman : Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv :1409.1556*, 2014.

[10] Hideyuki Tamura, Shunji Mori et Takashi Yamawaki : Textural features corresponding to visual perception. *IEEE Transactions on Systems, man, and cybernetics*, 8(6):460–473, 1978.

[11] Carles Ventura Royo : Image-based query by example using mpeg-7 visual descriptors. 2010.

[12] Vandana Vinayak et Sonika Jindal : Cbir system using color moment and color autocorrelogram with block truncation coding. *International Journal of Computer Applications*, 161(9):1–7, 2017.

[13] Chee Sun Won, Dong Kwon Park et Soo-Jun Park : Efficient use of mpeg-7 edge histogram descriptor. *ETRI journal*, 24(1):23–30, 2002.

[14] Konstantinos Zagoris, Savvas A Chatzichristofis, Nikos Papamarkos et Yiannis S Boutalis : Automatic image annotation and retrieval using the joint composite descriptor. *In 2010 14th Panhellenic Conference on Informatics*, pages 143–147. IEEE, 2010.