Proyecto Final de Aprendizaje de Máquina

Music Genre Classifier

Integrantes:

Leandro Rodríquez Llosa

Grupo: C-41X

Cuarto año. Ciencias de la Computación.

Facultad de Matemática y Computación, Universidad de La Habana, Cuba

Junio 2023

Resumen

TODO

Palabras clave — aprendizaje automático · clasificación · géneros musicales

I. Introducción

TODO

II. Propuesta

Hasta ahora hemos implementado algunas de las ideas más exitosas para clasificar géneros musicales que se pueden encontrar en el estado del arte. Pero resulta que ninguna de estas hace uso de la letra de la canción, la cuál sabemos intuitivamente que puede aportar bastante información sobre el género de la canción. Por tanto, una idea implementada en este trabajo es hacer un modelo clasificador que conste de una red neuronal que combina análisis de la letra, mediante un embedding que represente a la misma, y de la música con un encoder. La idea de este modelo es concatenar los vectores resultantes de estos procesamientos y clasificar en base a esta combinación de features.

1. Autoencoder

Para construir el encoder se programó un autoencoder y se tomó el modelo hasta el bottleneck para sacar el encoder. La arquitectura del autoencoder combinó capas maxpooling y upsampling al inicio y al final respectivamente para moderar el tamaño la imagen, intercaladas con capas convolucionales y en el medio tuvo un par de capas densas para aprovechar que ya el número de dimensiones era relativamente pequeño y realizar un poco más de aprendizaje. La arquitectura en detalle es la siguente:

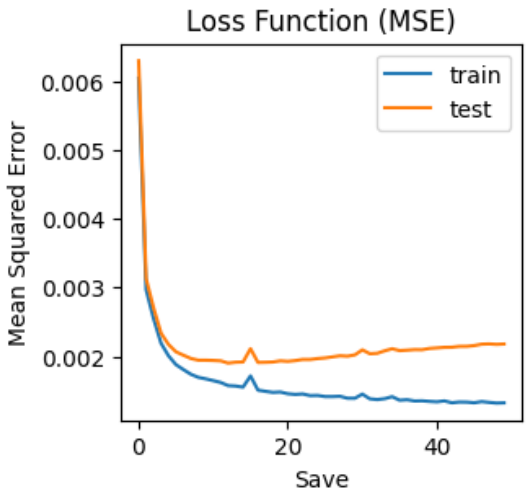
Nombre de la capa	Tipo de la capa	Shape de salida
$input_l$	Input	(192, 256, 3)
$maxp_{ini}$	MaxPooling2D	(96, 128, 3)
$encoding_1$	Conv2D	(96, 128, 12)
$maxp_1$	MaxPooling2D	(48, 64, 12)
$encoding_2$	Conv2D	(48, 64, 6)
$maxp_2$	MaxPooling2D	(24, 32, 6)
$encoding_3$	Conv2D	(24, 32, 3)
$flat_1$	Flatten	(2304)
$bottleneck$	Dense	(500)
$decoding_1$	Dense	(2304)
$resh_1$	Reshape	(24, 32, 3)
$decoding_2$	Conv2D	(24, 32, 6)
Up_1	UpSampling2D	(48, 64, 6)
$decoding_3$	Conv2D	(48, 64, 12)
Up_2	UpSampling2D	(96, 128, 12)
$decoding_4$	Conv2D	(96, 128, 3)
$output_l$	UpSampling2D	(192, 256, 3)

Como se puede observar la arquitectura es casi simétrica.

Se tomó como función de pérdida y métrica el error cuadrático medio (*mean squared error*). La entrada del autoencoder y la salida esperada fueron las imágenes del feature MFCC del conjunto de entrenamiento luego de haber sido

normalizadas, es decir, que en vez de estar en el rango $[0, 255]$ cada valor de la imagen de entrada, los representamos en el rango $[0, 1]$.

Anteriormente se probaron otras arquitecturas, desde algunas que no tenían capas densas hasta otras que principalmente consistían en capas densas. El problema en las arquitecturas carentes de capas densas era que sus resultados no eran lo suficientemente buenos, es decir, debido a su relativamente baja cantidad de parámetros el nivel de aprendizaje que podían lograr era inferior al que se logró luego con arquitecturas con capas densas. Por otro lado, las arquitecturas que consistían principalmente en capas densas tenían problemas como que los modelos eran muy grandes, algunos pasando de los GiB de almacenamiento y presentaban un problema para nosotros a la hora del entrenamiento. Otro problema que tienen las arquitecturas más basadas en capas densas es su tendencia al overfitting. En las prueba realizadas, las arquitecturas carentes de capas densas no presentaban este tipo de problema ya que los resultados en los conjuntos de entrenamiento, test y validación tenían poca diferencia entre ellos, sin embargo en las arquitecturas que tenían capas densas, por el gran número de parámetros si se evidencia una diferencia sustancial entre los resultados en los conjuntos de entrenamiento, y los obtenidos en los de prueba y validación. Al entrenar se realizó un *save* cada 10 epochs. Veamos los resultados sobre el número del *save* en el modelo de autoencoder presentado:



A partir del análisis de la gráfica anterior se tomó como cantidad de epochs a ejecutar la cantidad de 150, ya que se conjeturó (y luego validó) que el comportamiento del modelo en el conjunto de prueba sería muy similar al comportamiento en el conjunto de validación y en este punto es que se obtiene un mejor performance en el conjunto de prueba.

Volviendo atrás, los resultados obtenidos para cada tipo de modelo:

- los modelos que no tenían capas densas lograron primeramente un error (MSE) de 0,0025 y luego de ampliar la cantidad de dimensiones que salen del bottleneck, es decir la cantidad de dimensiones de la representación se logró 0,0021. Estos modelos tenían muy poco overfitting luego de 500 epochs.
- los modelos que presentan capas densas, por su parte, comenzaron con resultados en el conjunto de entrenamiento de hasta 0,0011 con 500 epochs, lo cual era muy bueno pero podía implicar overfitting. A medida que se redujeron la cantidad de parámetros (de 288 millones a los 2,3 millones del modelo propuesto) los resultados en el conjunto de entrenamiento fueron peores pero nunca sobrepasaron el valor de 0,0013 en 500 epochs. Luego de correr el modelo propuesto solo 150 epochs se obtuvieron los mejores resultados tanto en el conjunto de prueba como en el validación, oscilando alrededor de 0,00185, por su lado en el conjunto de entrenamiento se obtuvieron resultados alrededor de 0,0016, lo que evidencia la presencia de overfitting.

Se realizó también cross validation con Kfold dividiendo todo el conjunto de entrenamiento en 10 subconjuntos. Los resultados del cross validation coincidieron con los resultados obtenidos en el entrenamiento del modelo sobre el conjunto de entrenamiento y luego evaluado sobre el conjunto de validación. Este test se hizo luego de haber fijado la cantidad de epochs

en 150.

TODO

II. Feature: Letra de la canción

La segunda mitad de la información de la que queremos que nuestro modelo aprenda es la letra de la canción. Para ello lo primero que se hace es extraer la misma haciendo uso del conocido modelo de Whisper (ver [1]). Luego necesitamos un embedding que represente al texto de la canción que vamos a necesitar para entrenar y evaluar el modelo. Para extraer el mismo hacemos uso de otro modelo de aprendizaje de máquinas conocido como Bert (ver [2]). Este último es un modelo basado en redes neuronales para el procesamiento de lenguaje natural, desarrollado por Google.

III. Arquitectura del modelo

Una vez obtenidos los dos embedding correspondientes a la letra de la canción y la música (MFCC) ya tenemos la entrada de nuestra red neuronal. La misma será la concatenación de los dos vectores obtenidos. Luego las siguientes capas .....

arquitectura final

TODO

TODO

TODO

III. Tests

IV. Recomendaciones

Referencias

[1] A. Radford, J. Wook Kim, T. Xu, G. Brockman, C. McLeavey y I. Sutskever: *RobustSpeechRecognitionviaLarge-ScaleWeakSupervision*. Preprint on arXiv: 2212.04356, 2022.

[2] J. Devlin, M. Chang, K. Lee y K. Toutanova: *BERT:Pre-trainingofDeepBidirectionalTransformersfor LanguageUnderstanding*. Preprint on arXiv: 1810.04805, 2018.

[3] Kevin P. Murphy: *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.