Multi-label film genre inference using convolutional neural networks on film trailers

Hannes Kristján Hannesson
School of Computer Science
Reykjavík University
hanneskh15@ru.is

Sigurður Helgason
School of Computer Science
Reykjavík University
sigurdurhel15@ru.is

Abstract

This paper describes a process for the identification of multiple genres for film trailers. The task of multi-label genre identification is a difficult task, which we attempt to solve by applying state of the art machine learning methods for the creation of a deep neural network. We discuss what methods we use and how we implement them, as well as describe our results. The code is all open source and available to everyone.[2]

1 Introduction

Multimedia label inference is a task that is of interest to not only the authors, but information repositories such as The Internet Movie Database, and video hosting sites such as YouTube.

In this implementation a convolutional neural network was used, but as trailers can be considered time-series data we decided to model time with another dimension in the data, this can be considered as a *block* where each increment on the Z-axis is the following frame from the previous frame of the trailer. This allows for a simulation of time-series data.

Throughout out the process we slice these blocks of trailers into equally sized subblocks, this process is called *slicing* and is used extensively in this report. A pictorial example of this process can be seen in Figure 1 and a further discussion is provided in 3.2.

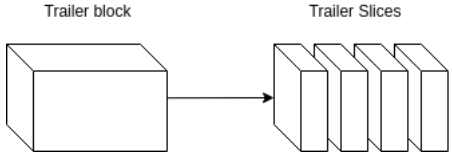

Figure 1: Slicing method applied to a trailer.

Using the slicing method, we implemented a deep convolutional neural network model for multi-label classification of movie genres.

2 Data

Data was gathered from the GroupLens Research website, which offered archived `csv` files containing film metadata from MovieLens, two of these `csv` files, namely `ml-youtube.csv`[4] and `movies.csv`[3], were used.

2.1 `ml-youtube.csv`

The structure of `ml-youtube.csv` consists of three columns, youtubeId (partial URL for the trailer link), movieId (ID of the movie in the MovieLens database), and title (title of the movie). A portion of the `csv` file can be seen in Table 1

2.2 `movies.csv`

The structure of `movies.csv` consists of three columns, movieId, title, and genres (the film genres

youtubeId	movieId	title
K26_sDKnvMU	1	Toy Story (1995)
3LPANjHlPxo	2	Jumanji (1995)

Table 1: Structure of `ml-youtube.csv`

listed in the MovieLens database). A portion of the `csv` file can be seen in Table 2.

2.3 `preprocessed_movies.csv`

We did a series of data preprocessing steps on the data provided by GroupLens, namely joining the two datasets by their movieId and constructing individual columns for each of the genres listed for the movies. The genres were 18 in total: Adventure, Animation, Children, Comedy, Fantasy, Romance, Drama, Action, Crime, Thriller, Horror, Mystery, Sci-Fi, Documentary, War, Musical, Western, and Film-Noir. These columns had a binary value 1 representing that the film was of that genre and 0 otherwise. Initially the distribution of classes in the dataset was very skewed towards drama. The models trained with the initial dataset were biased towards ascribing the genre drama to all movies, because of this steps were taken to decrease the gap between classes, namely removing all instances of films whose only genre was drama, and to remove all instances whose classes were both Drama and Comedy. The class distribution before and after these steps were taken can be seen in Appendix B. The final preprocessing step was to prune the instances in the dataset whose trailer was either too short, too long, or no longer available to us on youtube.

2.4 Training data

For training, we used video frames from trailers of 5637 films, we extrapolated a frame once every 100 frames, The resulting frames amounted to 30 GiB of data. The frames extracted were then resized to ($64\times$ 64)-pixel size, this was not only to reduce size but also to try to de-emphasize small details and emphasize the large details. An example of this can be seen in Figure 2. One can see that the larger details are still very visible, and the small details have been de-ephasized.

Figure 2: Image resizing for detail fuzzing. Left image is original, right image is resized image.

3 Methods

3.1 Data generator

Due to the size of the training data, storing the data in memory is infeasable. To circumvent this issue, we load data into memory as needed, this is done via a *generator*. This generator was implemented in accordance with the Keras [1] API, as Keras supports using generators natively and is the recommended way to handle this memory issue.

3.2 Slicing

The basic operation in training a neural network, matrix multiplication, requires that matrices be of the same dimensions. This is a cumbersome fact when our matrices are based on variable length film trailers, thus we must find a way to make sure all of our matrices be of the same size.

This is simple for the width and height dimensions, we simply resize the frames to be of some predetermined size. However, how we ensure

movieId	title	genres
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
2	Jumanji (1995)	Adventure|Children|Fantasy

Table 2: Structure of `movies.csv`

ID	youtubeId	movieId	title	Romance	...	Action	Thriller
0	K26_sDKnvMU	1	Toy Story (1995)	0	...	0	0
1	3LPANjHlPxo	2	Jumanji (1995)	0	...	0	0

Table 3: Structure of the combined `csv` file

that the depth (i.e. time) is invariant is a bit trickier.

A few solutions exist for this, one would be to just trim all the trailers to be of the same length as the shortest trailer, however, we lose a lot of info this way. Another solution would be to pad *zero-frames*, that is, a frame with RGB values $(0, 0, 0)$ for every pixel. This is what is done in this implementation, each slice receives a predetermined amount of frames, and if this amount is not divisible by the number of slices we are using, then the last slice is padded with zero-frames. The amount of zero padded slices is therefore minimized, and the zero padding has little effect on the predictions by the model.

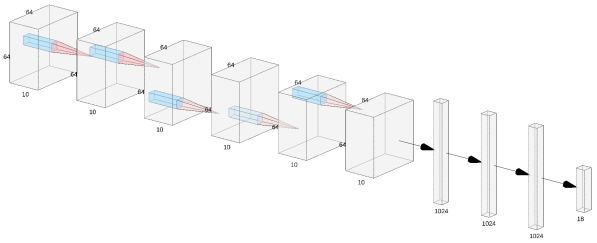

Figure 3: 3D visualization of the model architecture, the layers resembling blocks are the 3D convolutional layers, the tall and short layers are the dense layers, *Note*: Pooling and Normalization layers are not shown in this visualization

3.3 Model

A convolutional neural network (CNN), consisting of an input layer, followed by 6 convolutional layers, followed by 3 fully connected layers, finally followed by an output layer consisting of 18 nodes. On all hidden layers, a rectified linear unit (ReLU) was used as an activation function; While a sigmoid activation function was used on the output layer. A sigmoid activation function was used on the output since a confidence output for each class is desired.

The *Adam* optimization algorithm with the hyper-parameters Learning Rate and Decay set to 0.0001 and $1 \times e^{-6}$ respectively.

The loss function the model utilized was *Binary Cross-entropy*, as in multi-label classification a loss function where the loss of one output tensor doesn't affect the update of the other tensors is desired.

An overview of the Keras model can be seen in Figure 5 along with a visual representation in Figure 3

3.3.1 Limitations

For the CNN to function, a standardized input is required, this refers to the fact that all frames must be of the same dimensions and have the same number of color channels, likewise, the slices must be of the same size.

3.4 Accuracy Metrics

The accuracy metric used is categorical accuracy, although we are wary that this function is not the most applicable for this task, as categorical accuracy only interprets a single genre which the model is most con-

fident on.
We did not have time to implement a custom accuracy metric, as we would have to write it with TensorFlow API calls, which none of the authors have the experience with.

3.5 Threshold metric

We did implement an auxillary function for use on the test data results. The function, which we call the *threshold metric*, which given a lower and upper threshold, and a prediction from the model, gives a report of the set of genres for which the model is confident that the film belongs to. In our case if the model has a confidence above 0.75 that a film is of a given genre, that genre is a part of the set.

4 Results

Using the accuracy metric discussed in Section 3.4, we achieve $\sim 90\%$ accuracy, with the loss ending at ~ 0.3, both these metrics seem to plateau at these values.

Figure 4: Accuracy and loss on 10 epochs

The authors evaluated the model on a number of hand-picked films, the results can be seen in Figures 7, 8, and 9. Although we did not calculate recall nor f1-score, we suspect it to be quite bad based on the results on the hand-picked films we ran the model on.

5 Discussion

The results achieved with this implementation are unsurprisingly bad. We believe the large factors that decrease the models performance, is the amount of data that we had, as can be seen from the results the confidence levels of the models for each class closely resembles the initial class distribution of the data; which indicates that not enough data is trained on in order to find the corresponding underlying function for the neural network to learn.

There were some attempts to balance the data (see Appendix B) in hopes to get a better result, however this resulted in a lot of the genre confidences to be more-or-less the same, with value ~ 0.5.

A second large factor we believe is that we only took every 100th frame of each film trailer, this makes it such that each following frame within a trailer slice is less indicative of the previous one, therefore leading to less connected data, and therefore lessens the need for a 3D convolutional neural network.

The last factor is that for time series data like this a recurrent neural network with LSTM is better suited for the task.

6 Future Work

Using recurrent neural networks (RNNs) might be worth taking a look at for this task, as [5] had good results using RNNs.
Training using more frames per film, might be beneficial to the model and might be worth taking a look at.

7 Related Work

[5] used RNNs with LSTM in conjunction with other methods to achieve $\sim 85\%$ accuracy on a balanced dataset with similiar recall, although they only concern themselves with a single-label classification.

References

[1] François Chollet et al. *Keras*. https://keras.io. 2015.

[2] Hannes K Sigurður H. *Github repository for this project*. URL: https://github.com/HKH515/MovieTrailerML/tree/92ffc9afb6bc5d81b5a4b3bd899c1e92dae3f0ba (visited on 12/15/2018).

[3] GroupLens Research. *MovieLens 20M Dataset*. URL: http://files.grouplens.org/datasets/movielens/ml-20m.zip (visited on 12/09/2018).

[4] GroupLens Research. *MovieLens 20M YouTube Trailers Dataset*. URL: https://grouplens.org/datasets/movielens/20m-youtube/ (visited on 12/09/2018).

[5] KS Sivaraman and Gautam Somappa. "MovieScope: Movie trailer classification using deep neural networks". In: *Dept. of computer science, university of Virginia* (2016).

A Keras model implementation

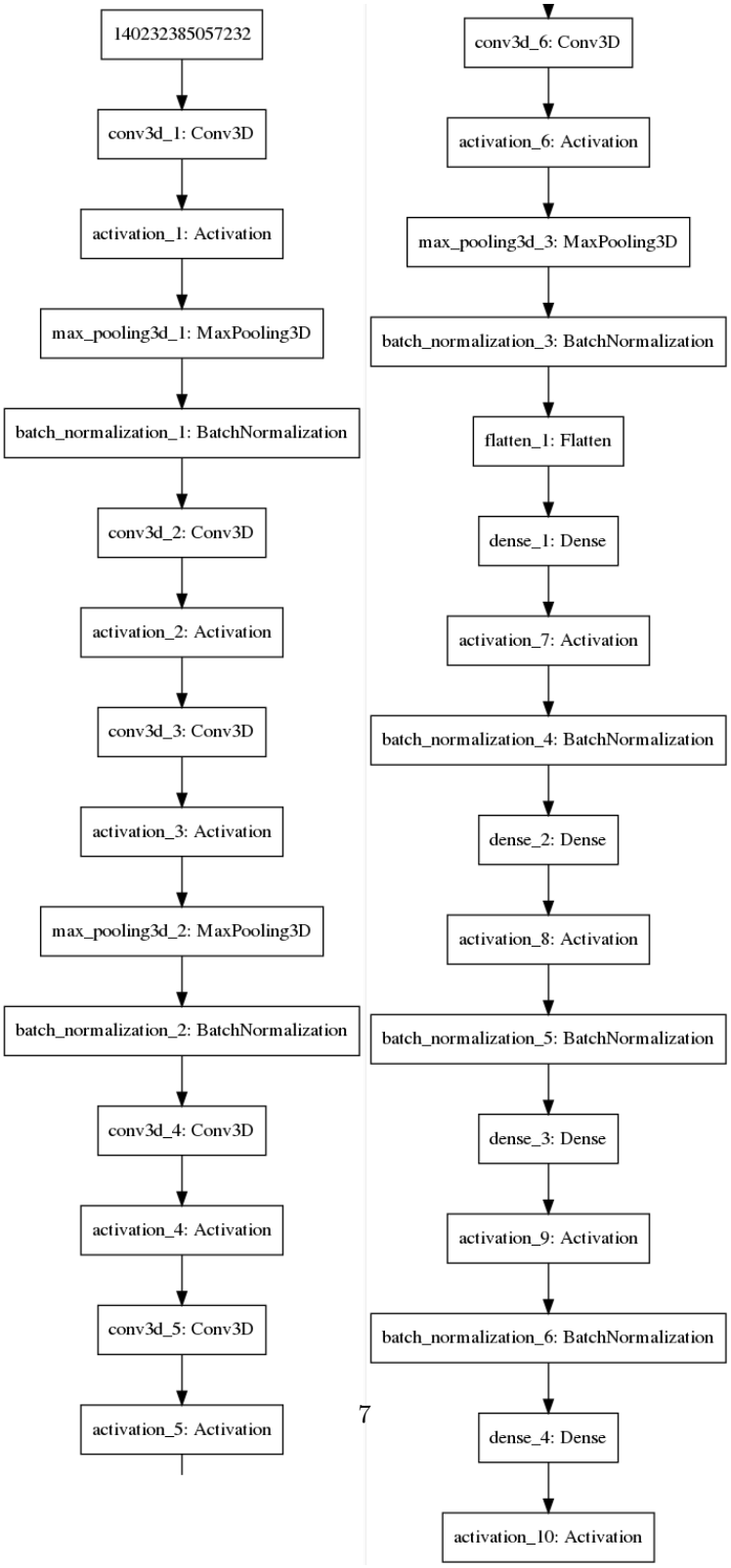

Figure 5: Keras CNN model

B Dataset distribution

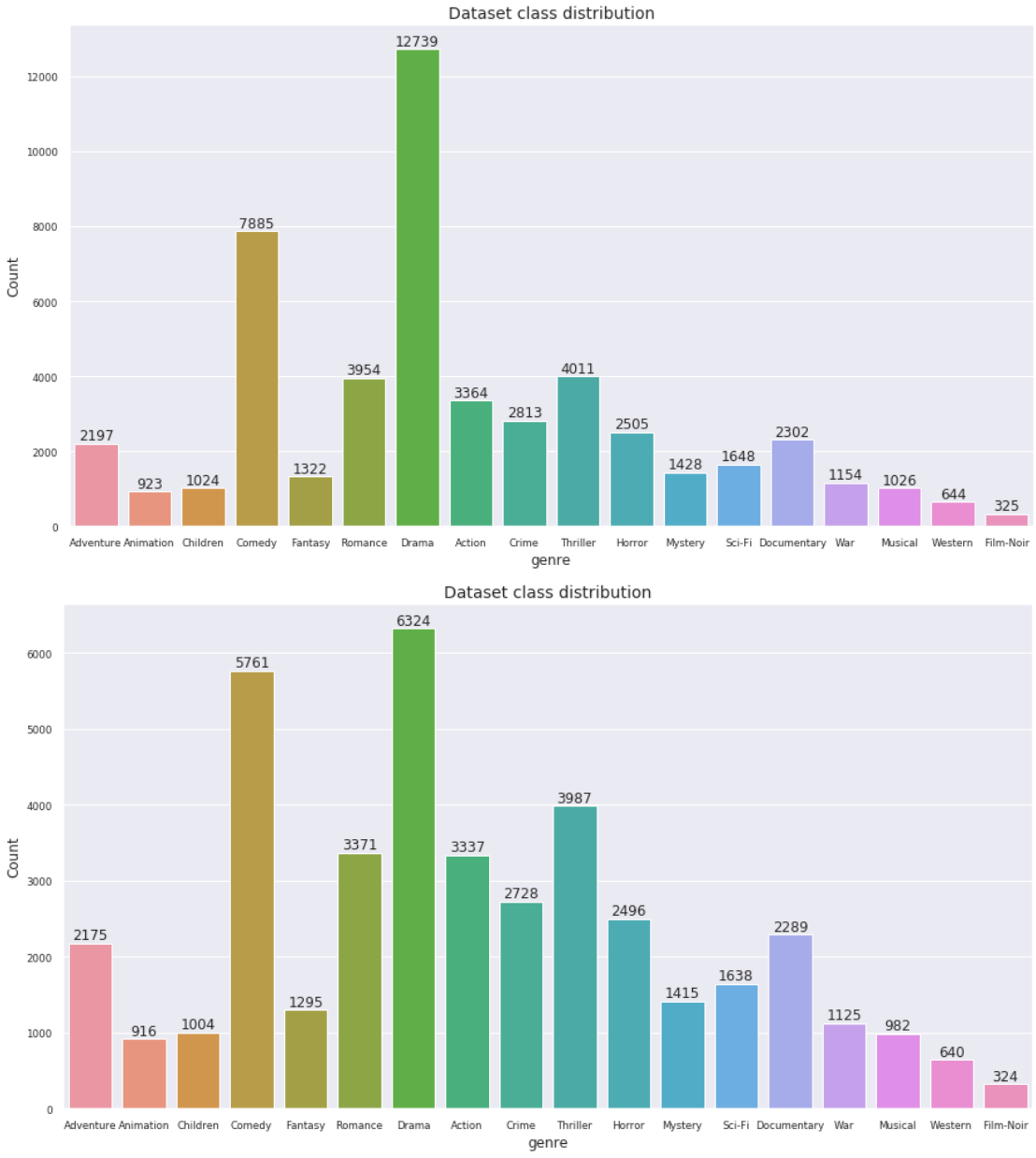

Figure 6: Class distribution before (above) and after (below) pruning the dataset.

C Model predictions for individual scenes

Figure 7: Frames from a scene from the Avengers: Endgame trailer, along with the model's predictions, the true genres are Action, Adventure, and Fantasy

Figure 8: Frames from a scene from the Avengers: Endgame trailer, along with the model's predictions, the true genres are Action, Adventure, and Fantasy

Figure 9: Frames from a scene from the Spider-Man: Into the Spider-Verse trailer, along with the model's predictions, the true genres are Animation, Action, and Adventure