Movie genre prediction using CNNs

Matteo Princisgh

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

1 Data preprocessing

The *letterboxd* dataset is composed of various `csv` files – of which only `genres.csv` was used for this work – and a folder containing all of the posters' images. This project was developed using a copy of the dataset downloaded the day 27/05/2024.

Each movie is identified by a numerical ID which is used both as an attribute in the `csv` file and as the name of the corresponding poster file. There are a total of 557˙539 movies for which both genre and poster are available, though only a subset of them was used, due to the fact that the genre distribution is heavily imbalanced, as shown in Fig. 1.

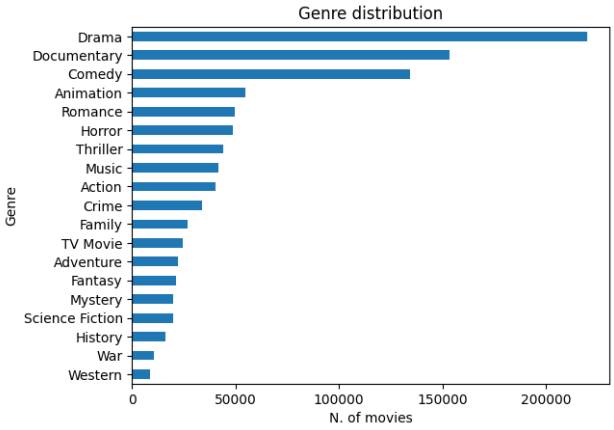

Figure 1: Label distribution

1.1 Label processing and balancing

Each movie may have more than one genre associated to it and, in order to simplify the subsequent steps (i.e. balancing and stratification) and to reduce the predictor's complexity, each multi-genre movie is collapsed into a single-genre one: given the genre distribution, if a movie has multiple ones associated to it, the least common is chosen as the movie's representative genre.

Balancing the labels simply requires identifying the least numerous class, in our case *Western* with 7822 movies, and randomly sample this number of movies from every other class. Given that there are 19 distinct genres in the dataset, the resulting number of data-points is 19 x 7822 = 148˙618.

Successively, data-points are split in the usual train/validation/test partitions, respectively using 70/15/15 splitting thresholds. Moreover, the three splits are stratified according to the balanced label distribution, so that each class' cardinality is homogeneous in every split.

Finally, labels are one-hot encoded using Scikitlearn's `LabelBinarizer`[1], which was chosen over `OneHotEncoder`, due to the fact that it allows for multi-label vector encoding.

1.2 Image loading and dataset creation

Once all the movie IDs and labels have been processed, the corresponding poster images are loaded and transformed using TensorFlow's `io` and `image` modules[2]: images are loaded and scaled to a resolution of 224x224 px, without changing their aspect ratio. This choice of resolution allows for a quick swap of the predictor's underlying CNN; it is in fact possible to use interchangeably either a ResNet [1] or an EfficientNet [2] as the backbone of the classifiers.

Finally, the `Dataset` objects are initialized, shuffled, and divided into mini-batches of size 128. By using TensorFlow's `Dataset` API, the loading of each batch can be deferred until the training phase. This allows the memory usage to be adjusted dynamically, regardless of the dataset's size.

2 Single CNN for multi-class classification

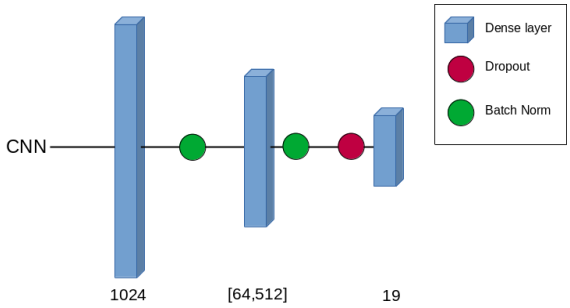

Figure 2: Structure of the head layers

The first classifier implemented is a single convolutional neural network that performs multi-class classification. As shown in Fig. 2, there is a 'tail' CNN – in this instance an EfficientNetV2B0 pre-trained on the *Imagenet* dataset – followed by a 'head' with three fully connected (FC) layers. The first two FC layers use the ReLU activation function, whereas the last one uses Softmax.

Early prototypes of this architecture exhibited a strong tendency of quickly overfitting in less than two epochs; this phenomenon has been mitigated using a combination of regularization techniques. Although combining batch normalization and dropout was traditionally discouraged, the results of [3] and [4] indicate that the choice between these techniques depends on the specific task the network is designed to solve. Therefore, a batch normalization layer is applied after each FC layer, and an optional dropout layer is placed before the output. The decision to enable the dropout layer is made during the model selection phase.

2.1 Model selection and training

In order to perform a moderate amount of model selection, the trainable hyperparameters are: the number of units of the second FC layer (within the $[64, 512]$ interval, with steps of 64 units), as well as the presence of the dropout layer. Due to time and hardware constraints, the model selection phase is kept at a bare minimum and performed using only a few steps of random search.

Subsequently, the model is compiled with the best hyperparameters found, then trained using the *adam* optimizer and the categorical cross-entropy loss function. The `EarlyStop` callback is used as the stopping criterion, ending the training phase in case the validation loss does not decrease for more than three epochs.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelBinarizer.html
[2] https://www.tensorflow.org/api_docs/python/tf

2.2 Results

The model performance is summarized in Tables 1 and 2; by analyzing the results for each class we observe that the model performs better on some classes rather than others. For example, it is sufficiently accurate in classifying Western movies, with an F-score of 0.52, whereas it performs poorly on Mystery ones, with an F-score of 0.10.

	Train	**Val.**	**Test**
Loss	2.31	2.42	2.42
Accuracy	0.27	0.25	0.25
Recall	0.06	0.06	0.06

Table 1: Performance of the model over the three splits

	Precision	**Recall**
Action	0.23	0.28
Adventure	0.18	0.11
Animation	0.32	0.53
Comedy	0.21	0.22
Crime	0.15	0.13
Documentary	0.18	0.24
Drama	0.15	0.11
Family	0.29	0.29
Fantasy	0.24	0.12
History	0.17	0.13
Horror	0.25	0.39
Music	0.36	0.19
Mystery	0.13	0.09
Romance	0.21	0.34
Sci-Fi	0.25	0.30
TV Movie	0.33	0.16
Thriller	0.18	0.12
War	0.29	0.35
Western	**0.46**	**0.60**

Table 2: Per-class performance of the model over the test set

3 One-vs-all CNN for multi-label classification

A One-vs-All (OvA) ensemble of binary classifiers was implemented to build a predictor with better per-class accuracy, which is also capable of performing multi-label classification. Each one of these binary predictors is structurally identical to the single CNN discussed above, apart from the output layer which is composed of a single unit with a sigmoid activation function.

3.1 Dataset creation and training

The binary classifiers require an ad-hoc dataset preprocessing step: each label vector in the parent dataset has to be reduced to a binary value, whose value depends on which classifier is going to be passed to. Moreover, these ad-hoc datasets are balanced in a 70/30 way (i.e. 70% of negative examples, 30% of positive ones), and stratified so that the negative examples are drawn uniformly across all of the 18 negative genres.

The training parameters remain the same as those used in the single CNN, with the exception of the loss function, which becomes binary cross-entropy. The stopping criterion for the training is, once again, early stopping with respect to the validation loss, with a patience of three epochs.

3.2 Results

As shown in Table 3, the per-class performance of each classifier is improved, compared to the results of the single CNN (Table 2). Additionally, the same OvA classifier can be used for both multi-label and multi-class classification, performing respectively a *thresholding* or an *argmax* of the output vector. In Table 4 there is a per-class comparison of F-scores achieved by the single CNN and the OvA computed over the same test set.

Independently of the model used, these overall poor scores are to be expected due to the high epistemic uncertainty of the task: unless there are some 'dead giveaways' in a movie's poster (e.g.,

cowboys for Westerns or a cartoonish style for Animation films), it would be incredibly difficult, even for a cinematography expert, to accurately predict the genre of a movie from its poster alone.

	Loss	**Accuracy**	**Recall**
Action	0.53	0.74	0.50
Adventure	0.57	0.71	0.48
Animation	0.37	0.83	0.72
Comedy	0.55	0.71	0.51
Crime	0.55	0.68	0.52
Documentary	0.52	0.75	0.51
Drama	0.57	0.69	0.51
Family	0.50	0.76	0.50
Fantasy	0.58	0.70	0.46
History	0.57	0.70	0.37
Horror	0.49	0.77	0.59
Music	0.53	0.73	0.46
Mystery	0.59	0.70	0.34
Romance	0.52	0.75	0.44
Sci-Fi	0.54	0.74	0.44
TV Mov.	0.56	0.72	0.48
Thriller	0.55	0.72	0.49
War	0.55	0.73	0.53
Western	0.36	0.85	0.71

Table 3: Performance of each binary classifier of the OvA

	F1 (OvA)	**F1 (CNN)**
Action	0.25	0.25
Adventure	0.11	0.14
Animation	0.39	0.40
Comedy	0.19	0.21
Crime	0.11	0.14
Documentary	0.21	0.20
Drama	0.13	0.13
Family	0.27	0.29
Fantasy	0.15	0.15
History	0.15	0.15
Horror	0.30	0.31
Music	0.28	0.25
Mystery	0.11	0.10
Romance	0.26	0.26
Sci-Fi	0.22	0.27
TV Mov.	0.27	0.21
Thriller	0.13	0.15
War	0.31	0.32
Western	0.51	0.52

Table 4: F1-score comparison of the two predictors over the test set

4 Efficiency and scaling

The pipeline implementation proposed in this project is robust with respect to the dataset size, regarding both memory footprint and execution time.

4.1 Memory usage

During the preprocessing phase, the `genres.csv` file is stored into a Pandas' `DataFrame`, which is loaded entirely in memory, but, since the file stores only the ID→Genre relation and since the number of movies produced in the world is limited, it is safe to assume that it will never grow to unmanageable sizes. In fact storing a million relations requires approximately 15MB and the 148˙618 data-points used in this work occupy only 2.38MB in memory.

Posters' images are loaded and processed lazily by TensorFlow's `DataSet` APIs, which manages dynamically both the prefetching and deallocation policies for the batches of data. The results reported above were achieved using a batch size of 128, which roughly translates to 20MB of RAM occupancy per-batch.

Finally, each model requires approximately 30MB of RAM to store its parameters and, in case of the OvA classifier, the memory footprint is directly proportional to the number of unique genres in the dataset.

4.2 Time complexity

The theoretical time complexity of the preprocessing phase is $\mathcal{O}(n \log n)$, where n is the number of records in the `genres.csv` file; this is quickly verifiable because there are a few calls to the `sort()` method, whereas the rest of the operations performed over the genres' dataframe are linear in time.

However, the time needed to load an image from filesystem to RAM and the time needed for a model training/inference step, is orders of magnitude bigger than the time required for a generic operation on a dataframe record. Thus, in practice, the pipeline execution time follows a linear growth, with respect to the number of posters in the dataset.

There are two additional optimizations that could be implemented to speed up the pipeline:

- **OvA classifier:** since every binary predictor is totally independent from the others, both training and deployment may be performed in parallel on multiple GPUs or in a distributed fashion across multiple nodes.
- **Model selection:** since every trial of the model selection phase is independent, each retraining iteration may be distributed across multiple nodes. Depending on the hardware capabilities, it is possible to increment the number of steps in the hyperparameter optimization phase, as well as using more refined optimization algorithms (e.g. Hyperband [5]).

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[2] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, 2021.

[3] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019.

[4] Bum Jun Kim, Hyeyeon Choi, Hyeonah Jang, Donggeon Lee, and Sang Woo Kim. How to use dropout correctly on residual networks with batch normalization. In *Uncertainty in Artificial Intelligence*, 2023.

[5] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 2018.