A Scalable Deep Learning Solution for Multiclass Classification of Book Covers

Final project for the course *Algorithms for Massive Datasets*

Ivan Mazzon

September 2025

Declaration
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 offense 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. No generative AI tool has been used to write the code or the report content.

1 Introduction

Convolutional neural networks (CNN) are a deep learning class of models that achieve the state-of-the-art results in the field of image classification. As the volume of available data has grown significantly in the recent history, the scalability of these models has become increasingly crucial.

Placing emphasis on this last characteristic, this project proposes a methodology to define, train and evaluate a CNN model, with the goal of classify books by category, given the relative cover image.

2 Dataset analysis and preprocessing

The dataset considered in this project is Amazon Book Reviews dataset [1]. It is made up of two files that contain respectively books data and book reviews. For the purpose of the project, only the former is taken in account.

The file contains approximately 150.000 full defined entries in which each book is associated to a category. There are more than 7.000 categories. The analysis of cumulative sorted frequencies of the categories shows that they are highly unbalanced. Indeed, the number of members of the largest 11 categories is nearly equal to the number of members of the other ones. Moreover, most of the categories are of little significance due to their spurious meaning or their small membership base. For instance, the novel *Fahrenheit 451* belongs to the *book burning* category (and, luckily, it is the only member).

In the light of these considerations, only the categories that have more than 2200 examples are considered — i.e. the 12 largest ones — and it was chosen to pick that number of examples from each category to have a balanced dataset that is more suitable to train a deep learning model. Hereafter, this subset of instances will be referred to as the *dataset*.

2.1 Data retrieving

Since the dataset contains only the URL of cover images, a retrieving method is required. Leveraging Python's `ThreadPoolExecutor` [2] images can be downloaded concurrently, significantly improving performance in terms of time. The images are manipulated and stored in two folders in mass storage, `train` and `test`, to distinguish their function in the training procedure, each one with a dedicated folder for each class. Given this organization, the Keras' function `image_dataset_from_directory` allows to easily set the source of both training and test sets to create the relative Tensorflow Datasets.

2.2 Data format

The images referenced in the dataset do not have a standard format, but the data must be uniform to be suitable for feeding a deep learning model. Thus, during the downloading phase, all the images are resized to have the reduced dimension 100×150, maintaining the common ratio between height and width of covers, that is about 1.5. In this way, the shapes of elements and fonts inside the images are almost preserved. Then, images are saved in `.jpg` format with RGB channels of colors. A normalization of color channels is performed during the loading of the Tensorflow Dataset instances using the `map` function. The labels are encoded from textual names of categories to one-hot vector representation.

3 Model

3.1 Model definition

The model is made up of the following components: four convolutional layers with ReLU activation function interleaved by batch normalization and max pooling layers; a flattening layer; a dropout layer; two hidden dense layers with ReLU activation function interleaved by as many dropout layers; a dense layer with a number of neurons equals to the number of classes ends the model.

The softmax activation function is chosen to interpret the results as the probability distribution of likelihood of an instance to be member of a class. In convolutional and dense layers is applied the L2 kernel regularizer. The adopted optimizer is Adam and, since we are operating with labels with one-hot representation, the chosen loss function is the categorical cross-entropy.

3.2 Model selection

In order to tune the hyperparameters for model selection, a grid search is performed by using the class `GridSearch` of Keras Tuner [3]. Fixing a subset of values a priori for selected hyperparameters, grid search is an exhaustive evaluation of the model with every combination of hyperparameter values. For each configuration, the model is trained for a small number of epochs using the training set and evaluated with the validation set. The hyperparameter combination that obtain the optimal performance metric — in the case of this project, the accuracy — will be used to implement the definitive model.

The grid search process has the limitation to be computationally expensive due to the "brute-force" evaluation. In order to optimize this process, early stopping is introduced. This method simply allows to skip to the next hyperparameters values combination if a given metric does not have the expected behavior for a certain number of consecutive epochs. In this project, early stopping is applied to grid search considering the validation loss, skipping when it does not decrease.

For hyperparameter tuning there are many degrees of freedom. In this project, to the aim of contain execution time, only a small subset of hyperparamers are considered for model selection.

4 Solution scalability

This section analyze the proposed solution in term of scalability, showing the implemented techniques, introduced to cope with large scale datasets.

4.1 Data lazy loading and prefetching

As mentioned above, the Tensorflow Data API [4] is used to create the `Dataset` objects needed to feed the deep network. This choice offers one main advantage: given the source of data, the dataset instance actually loads the data only when it is requested. This lazy approach permits to avoid storing the entire dataset in main memory, giving the possibility to handle large datasets loading only the needed fraction at a time.

Besides, the prefetching technique [5] is applied to the dataset. This allows to load and preprocess in advance the mini-batches required for the next training step while the current one is running. The process is handled by a concurrent background thread and an internal buffer in which data is loaded ahead the time is requested. By overlapping data production and consumption, the usage of CPU and GPU is maximized, with the latter not having to wait the loading of data for the next step. This leads to a remarkable reduction of execution time for each epoch during the training of the model.

4.2 Hardware accelerators

Training a deep learning model without an hardware accelerator, e.g. the GPU, is clearly unfeasible in terms of time, especially when we are dealing with large datasets. Furthermore, the use of GPUs is justified by the fact that convolutional layers computations are highly optimized on those architectures.

There is also the possibility to use multiple GPUs in parallel. This approach enables the training of multiple models at the same time. This is particularly useful in model selection since it is possible to run the same model with different hyperparameter values in parallel [6].

Another way to exploit this hardware is training a single model on multiple GPU through data parallelism. The model is replicated on each device and the training step is executed simultaneously. Tensorflow implements a kind of data parallelism, the mirrored strategy, that consists in replicate all the paramters across the GPUs and apply the same parameter updates on every device [6].

Due to the limitations imposed by the Google Colab virtual environment, in this project the single-device approach is adopted, but it is easily adaptable to handle multiple GPUs setting the `tf.distribute.MirroredStrategy`.

4.3 Mixed precision

The mixed precision policy [7] is a quantization technique that permits to represent weights, activation and gradient values in half precision format (16 bits). However, a master copy of weights in single precision (32 bits) is maintained and updated.

Adopting mixed precision leads to several advantages. Firstly, training data requires half the space in device memory. This enables the possibility to load larger mini-batches, optimizing the use of the GPU and improving training efficiency.

Another advantage lies in modern GPUs that have hardware units specialized to perform half precision operations [8, 9], allowing a significantly faster model training. Finally, it has been empirically observed that, in many cases, models trained with mixed precision do not incur accuracy loss with respect to single precision baselines [7].

4.4 Checkpoints and fault tolerance

Since, although the optimizations, training computations on massive datasets may require a large amount of time, it is necessary to save intermediate results and make the procedure fault tolerant. These features are achieved using `ModelCheckpoint` and `BackupAndRestore` callbacks during the training phase.

The `ModelCheckpoint` callback saves the architecture and the weights of the best model obtained during the training according to a specific monitored metric. In this way, the full optimal model can be loaded at a later time, possibly to continue the training from the saved state.

The `BackupAndRestore` callback saves a backup of training state in a temporary file with a given frequency. If an interruption occurs during the process, the training state is restored loading the last backup, avoiding restarting of the entire computation.

5 Experimental setting and results

As discussed above, due to the hardware limitations and to the aim of contain time execution, it was chosen to extract 2200 examples from each of the 12 major classes. The dataset is partitioned in three subsets: a training set, containing 60% of the instances, a validation set, with 20%, and a test set, with the remaining 20%. It is ensured that the frequencies of the categories are almost balanced across the subsets. The mini-batch size is set to 128. The training and the validation set are used to feed and evaluate the model during the model selection. Then, they are used to retrain the optimal model found in order to monitoring its generalization capability. Lastly, the model is evaluated computing accuracy score on the test set.

As already mentioned, a subset of hyperparameters and their values are considered for model selection and these are the learning rate and the number of filters of convolutional layers. The hyperparameters selected for the grid search and their possible values are shown in Table 1. The grid search determined that the optimal values for the two hyperparameters are the learning rate and the number of convolutional filters. It was chosen to fix the number of filters of the first convolutional layer and increase by 16 that number in the following ones.

After the definitive training phase, the model reach an accuracy on the test set of 0.258. The evolution of accuracy and loss on training and validation sets over epochs is shown in Fig. 1. The charts reveal that the model is affected by overfitting. The model learn effectively from the training set but it cannot generalize well on unseen data.

The confusion matrix of predictions on a sample the test set consisting of 100 elements of each class is reported in Fig. 2. Despite the general presence of misclassified examples due to the low accuracy, the model can recognize some classes more clearly than others, such as *Computers*, *Juvenile Fiction*, or *Cooking*. There is also an evident confusion between *Juvenile Fiction* and *Juvenile Nonfiction*, probably due to the similar target of the books which might translate in similar cover arts. The worst recognized categories are *History* and *Social Science* that are mainly scattered in the other categories.

Some causes of this behavior can be an excessive complexity of the model; the need for a more advanced preprocessing; the need for a deeper tuning of hyperparameters; or categories, as they are defined, can be intrinsically barely distinguishable by cover arts.

Some examples of predictions are reported at the end of the notebook.

Table 1: hyperparameters values grid

Hyperparameters	Tuning values
n. of filters of the 1st convolutional layer	24, 32, 40
learning rate	$10^{-4}, 5 \cdot 10^{-4}, 10^{-3}$

Figure 1: accuracy and loss on training and validation sets over the epochs.

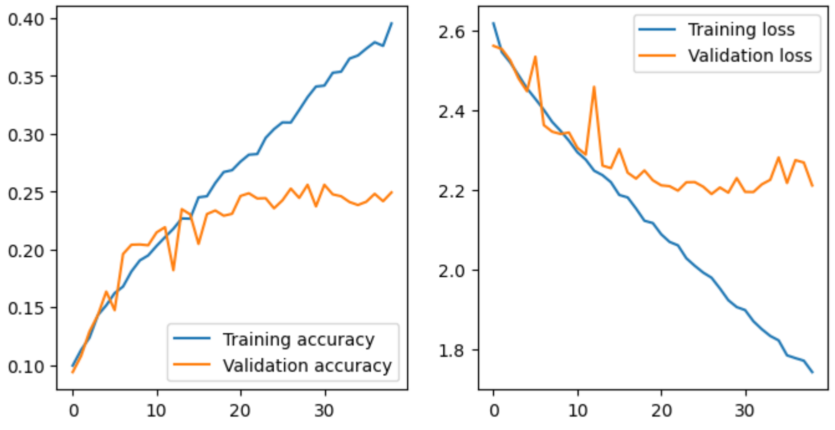
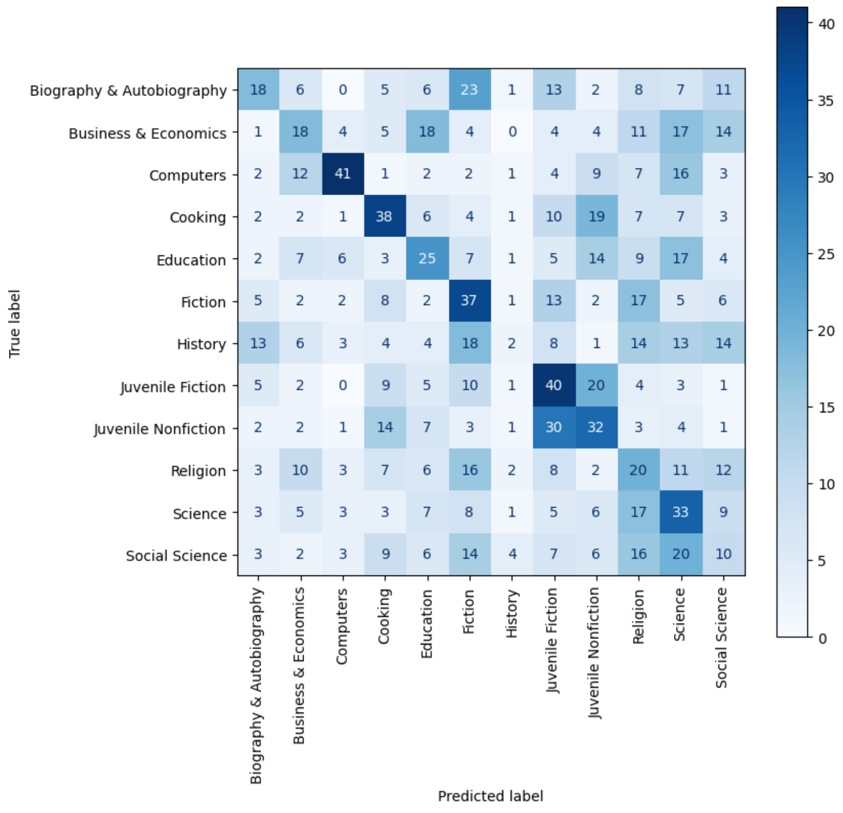


Figure 2: confusion matrix of predictions over a sample of the test set of 100 instances per class.



References

[1] Mohamed Bekheet. *Amazon Books Reviews.* URL: `https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews`. (accessed: 10.07.2025).

[2] *concurrent.futures — Launching parallel tasks.* URL: `https://docs.python.org/3/library/concurrent.futures.html#concurrent.futures.ThreadPoolExecutor`. (accessed: 31.07.2025).

[3] Keras. *Keras Tuner.* URL: `https://github.com/keras-team/keras-tuner`. (accessed: 21.07.2025).

[4] *Dataset API.* URL: `https://www.tensorflow.org/api_docs/python/tf/data/Dataset`. (accessed: 21.07.2025).

[5] Tensorflow. *Better performance with the tf.data API.* URL: `https://www.tensorflow.org/guide/data_performance`. (accessed: 21.07.2025).

[6] Aurélien Géron. "Training and Deploying TensorFlow Models at Scale". In: *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems.* 3. ed. O'Reilly, 2022. Chap. 19. ISBN: 9781098125974.

[7] Paulius Micikevicius et al. "Mixed Precision Training". eng. In: *arXiv.org* (2018). ISSN: 2331-8422.

[8] Tensorflow. *Mixed precision.* URL: `https://www.tensorflow.org/guide/mixed_precision`. (accessed: 04.08.2025).

[9] NVIDIA Deep Learning Performance. *Train With Mixed Precision.* URL: `https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html`. (accessed: 04.08.2025).