Image-Sentence Pair Matching

Sasu Alexandru-Cristian

January 19, 2025

1 General Observations

This project was carried out as part of this Kaggle competition for a binary classification task.

Right from the start, it could be observed that the train and validation datasets were balanced:

Label	Train	Validation
0	5000	1500
1	5000	1500

Table 1: Train and validation datasets samples distribution.

As such, there was no need to use metrics for imbalanced datasets (such as F1 score).

Given the small quantity of samples and their low number of features (especially the short length of sentences, at most approximately 50 words), training deep learning models proved to be a difficult task, as they were prone to overfitting. As such, training CNNs, LSTMs, or RNNs resulted in obtaining results similar to the following:

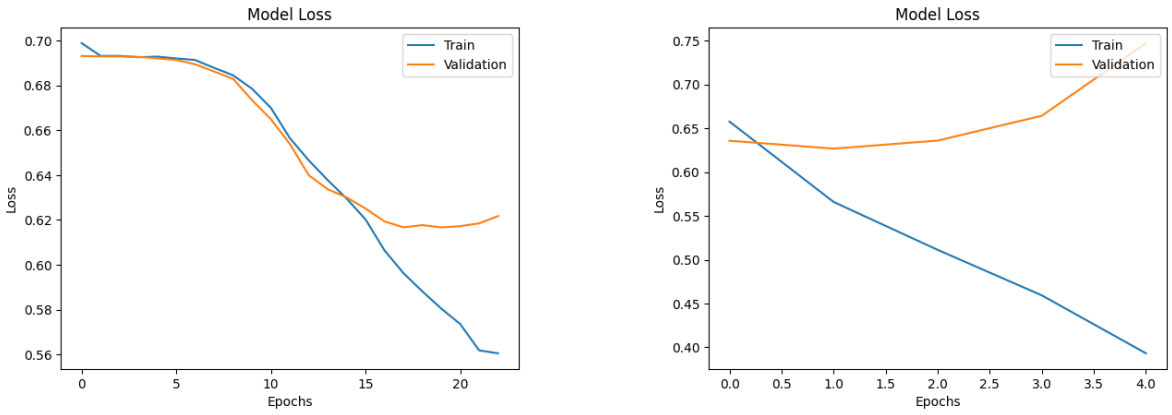

Figure 1: Overfitting examples.

Moreover, the test dataset had a lot of noisy samples where words had additional random letters or were missing letters thus not making sense. These samples were manually corrected. And, unlike the train and validation datasets, the test dataset had a more descriptive text samples, making it noticeably harder for models to generalize.

2 Data Preprocessing

In order to preprocess the text part of each sample, firstly some simple techniques were applied, namely: text to lowercase, punctuation removal, numbers removal, stop words removal. Afterwards, a Tensorflow tokenizer was trained from scratch on the preprocessed texts from the train dataset, and was applied on the texts from all datasets in order to obtain arrays of integer numbers of fixed length 60 (the number 60 was chosen because no text from the train dataset had more than approximately 50 words).

In order to preprocess the image part of each sample, the basic normalization operation of dividing each pixel by 255.0 was applied.

3 Model Strategies

3.1 Strategy 1: CNN and LSTM

This strategy obtained an accuracy of 46.75% on the public leaderboard.

The first strategy consisted of utilizing a model made up of two subnets, each with their own task. Thus, one of the subnets was a CNN used for image processing, while the other subnet was an LSTM used for text processing. After getting the processed features out of each subnet, the features were then concatenated and fed to a fully connected layer before passing through a sigmoid.

The LSTM subnet had an embedding layer trained from scratch and produced embeddings of length 50. The entire model was heavily regularized through the use of Dropout layers, due to the findings in section General Observations (80% neuron deactivation after each Dense layer, and 25% neuron deactivation after each Conv2D and LSTM layer). Moreover, the model used the Adam optimizer with a constant learning rate of 10^{-4}, along the binary cross-entropy loss function. Also, the model was trained for a maximum of 30 epochs and early stopping was employed, with a patience value of 3, that monitored the validation loss and restored the model with the best such value, and also a batch size of 32 was used during the training process.

More details about the model architecture and hyperparameters:

- CNN: used two Conv2D layers (the first had 32 filters and the second 64), each with a kernel of size 3 x 3 and the ReLU activation function, after each Conv2D layer, a MaxPooling2D layer with a kernel of size 2 x 2 followed, then came the 25% Dropout layers, and at last the Flatten layer. Then, the features passed through a Dense layer with 64 units and the ReLU function and then the Dropout layer with 80% neuron deactivation
- LSTM: used a single LSTM layer, with 32 units, then came the 25% Dropout layer, then a Dense layer with 64 units and the ReLU function, and then the 80% Dropout layer
- Combined subnets: used a single Dense layer with 64 units and the ReLU function, followed by the 80% Dropout layer, and the sigmoid layer

The results obtained for this approach were similar to the following:

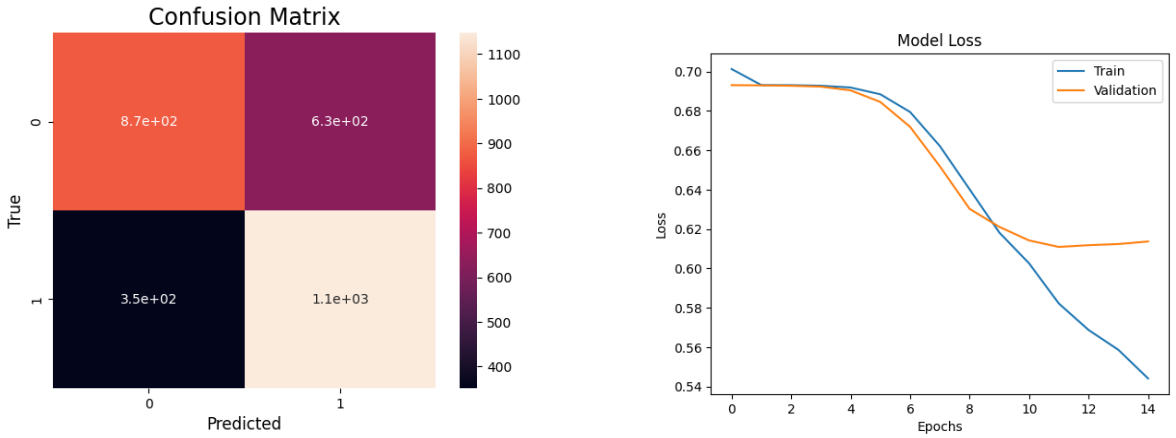

Figure 2: Confusion matrix for the validation dataset and loss curves for the first strategy.

The model obtained a 67% accuracy on the validation dataset, although some overfitting can still be observed from the loss curves, most probably given by the complexity of the network.

3.2 Strategy 2: LSTM

This strategy obtained an accuracy of 48.25% on the public leaderboard.

During further experiments, it has been observed that feeding both the given train images and texts to a model having a subnetwork for image classification and a subnetwork for text classification gives results similar to using just the text data as training data, along with the same subnetwork for text classification. As such, this strategy consisted of using only the LSTM from the previous strategy.

The model's architecture and hyperparameters remain the same as in the previous strategy.

The results obtained for this approach were similar to the following:

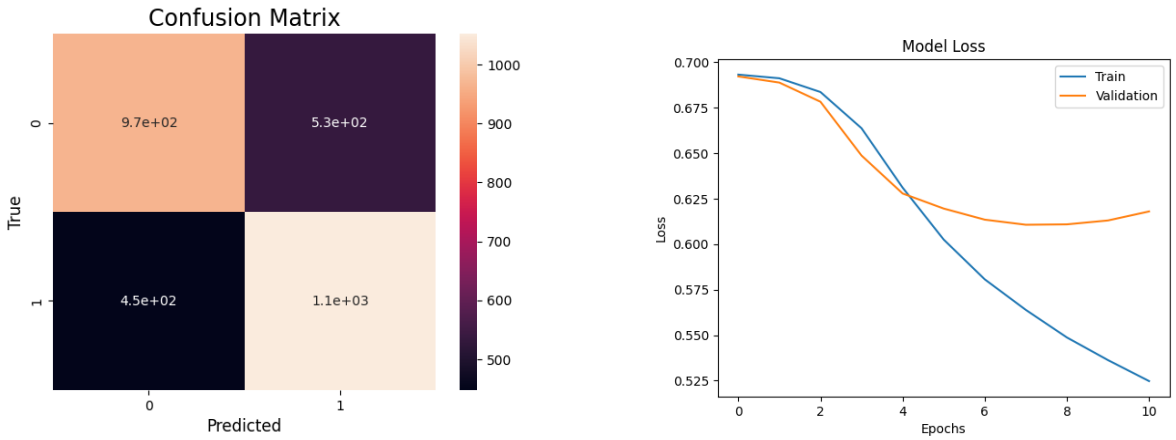

Figure 3: Confusion matrix for the validation dataset and loss curves for the second strategy.

Overall, this approach obtained mostly indistinguishable results from the previous one. The model obtained the same 67% accuracy on the validation dataset as in the previous strategy, while preserving a similar amount of overfitting.

3.3 Strategy 3: Strategy 1 + Random Forest

This strategy obtained an accuracy of 48.25% on the public leaderboard.

For this strategy, the model from the first strategy consisting of a CNN and an LSTM was also used, but this time for the purpose of extracting features. After obtaining the features, they were fed into a Random Forest model after finding suitable parameters for it during a grid search approach.

Following a PCA analysis of the validation features created by the deep learning model, it can be theorized that the model efficiently separated the samples in a linear manner:

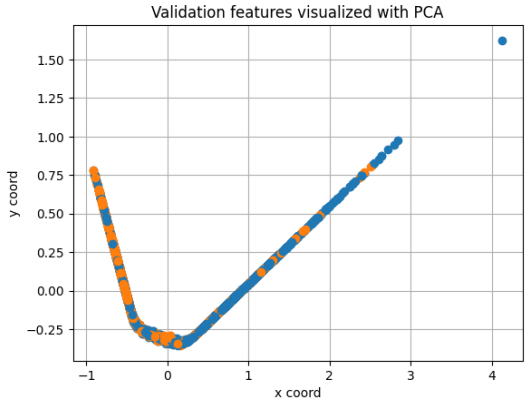

Figure 4: PCA analysis of validation features extracted by the CNN + LSTM model.

Random Forest grid search parameters:

- n_estimators: 50, 100
- max_features: 'sqrt', None
- max_depth: 5, 10
- min_samples_split: 2, 5, 10
- min_samples_leaf: 2, 4, 6

Random Forest first 5 and last 5 hyperparameter configurations, sorted descending by validation accuracy:

Hyperparameters	Train acc.	Validation acc.
50, 'sqrt', 10, 10, 6	0.7963	0.6856
100, None, 10, 10, 4	0.8088	0.6843
100, None, 10, 10, 6	0.8055	0.6843
50, 'sqrt', 10, 10, 4	0.7996	0.6840
50, None, 10, 2, 2	0.8097	0.6833
...	...	...
50, 'sqrt', 5, 5, 4	0.7733	0.6723
100, 'sqrt', 5, 2, 4	0.7712	0.6723
50, 'sqrt', 5, 2, 6	0.7713	0.6716
50, 'sqrt', 5, 10, 2	0.7729	0.6716
100, 'sqrt', 5, 10, 6	0.7717	0.6716

Table 2: Random Forest hyperparameter tuning.

From the above table, it can be observed that no matter the hyperparameters selection, the model obtains roughly the same performance on the validation dataset. However, the first combination has been chosen to train the final model.

The results obtained for this approach were similar to the following:

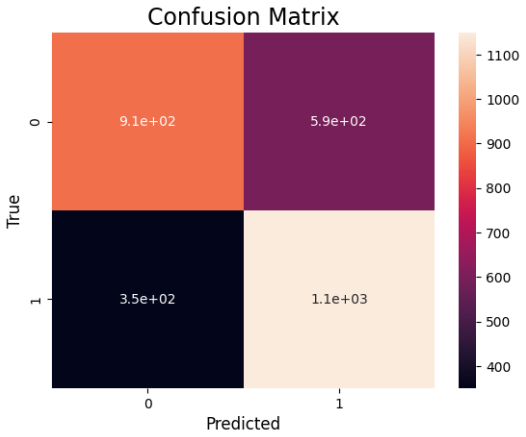

Figure 5: Confusion matrix for the validation dataset for the third strategy.

3.4 Strategy 4: Strategy 1 Revisited

This strategy obtained an accuracy of 53.75% on the public leaderboard.

As all three previous strategies obtained very similar results on both the validation and test datasets, it was decided to revisit just the first strategy in hopes of achieving better performance. As such, the model had small improvements (more specifically only the image classification subnetwork): added batch normalization after each Conv2D layer, and padded the images between these layers (using the "padding" hyperparameter from Conv2D); and also some aspects of preprocessing were changed: images were turned into grayscale, while text preprocessing was completely removed due to the already short sentences (meaning numbers, punctuation, and stop words were no longer removed).

The results obtained for this approach were similar to the following:

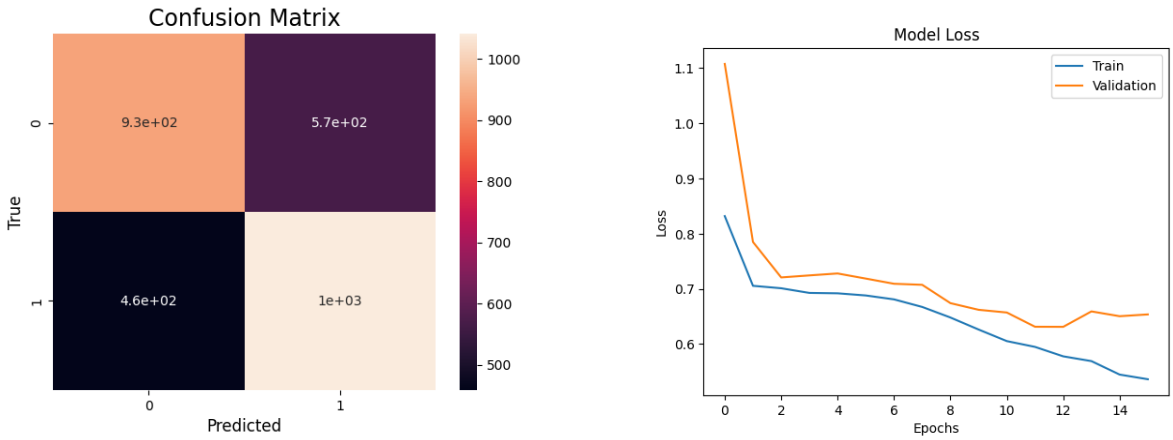

Figure 6: Confusion matrix for the validation dataset and loss curves for the fourth strategy.

Furthermore, a more detailed classification report was put together in order to try and understand why this atypical strategy obtained better results on the public leaderboard than all the other strategies:

	precision	recall	f1-score	support
0	0.77	0.41	0.54	1500
1	0.60	0.88	0.71	1500
accuracy	-	-	0.65	3000
macro avg	0.69	0.65	0.62	3000
weighted avg	0.69	0.65	0.62	3000

Table 3: Classification report for the fourth model strategy.

From Table 3, it may be extracted that the model obtained the presented results simply by correctly classifying more samples with label 1, in the detriment of misclassifying less samples with label 0.

The model obtained a 66% accuracy on the validation dataset, and noticeably different loss curves from the previous strategies (most likely due to the batch normalization layers that added additional regularization).

It is unclear how making the images grayscale and modifying the model architecture resulted in obtaining better results on the public leaderboard, as grayscaling should have reduced the number of meaningful features, and the model had an already complex enough architecture given the results on the validation dataset (around 67%) and the present overfitting factor. Though without any of the three new additions: modified architecture, images to grayscale, removal of text preprocessing; the model was not able to surpass 49.5% accuracy on the public leaderboard.

4 Conclusions

No matter the strategy used, the results obtained on both the validation and the test datasets were very close to each other between strategies (except for strategy no. 4), and the results on the test dataset were really poor. This points to the fact that the train and validation datasets are not comprehensive enough for the model to generalize well on the given test set.

Moreover, upon further inspection of the test dataset, it has been noticed that random noise has been purposefully added to the texts, as well as biasing factors (on average, the same text repeats considerably more than in the train and validation datasets; the test dataset only has 15 different texts, but 2000 samples). Furthermore, the text samples were noticeably more descriptive than in the other datasets. Given these aspects, it is not surprising that the models had a difficult time classifying the test samples.