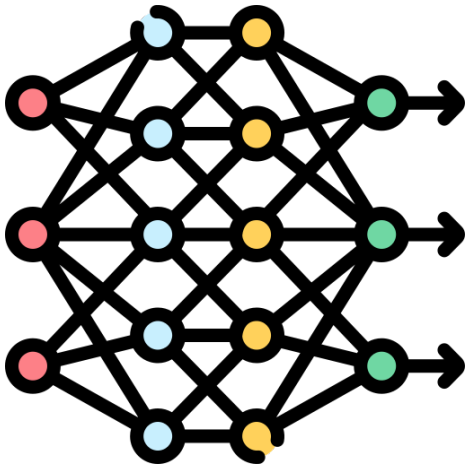
Deep Learning Project

Daniele Avolio **Alessandro Fazio** **Michele Vitale**



Contents

1 Introduction

Objective of the project is to compete in the Kaggle competition *LLM - Detect AI Generated Text* , which has the goal to develop a machine learning model to detect wether the input text has been written by a student or generated via the usage of a Large Language Model (such as ChatGPT,Bard, Claude).
Our focus was primarily on deep learning techniques, as it is the main topic of the course.

For further details and about the competition, please refer to the competition link: Kaggle.
Development of this project has been really dynamic, so that many different versions of every architecture has been tested out. This report should be intended as a non-exaustive, generic overview of the proposed architectures and a collection of the models that performed better in the Kaggle competition leaderboard.

2 Data understanding and preprocessing

2.1 Dataset Description

In this competition, we are given a **dataset** of **essays** written by **middle and high school students** and **large language models** (LLMs). The dataset is composed by 2 files:

- train_prompts.csv
- train_essay.csv

train_prompts.csv: Is a file that contains the questions that the students and LLMs answered. In particular, the structure is:

Field	**Description**
prompt_id	A unique identifier for each prompt.
prompt_name	The title of the prompt.
instructions	The instructions given to students.
source_text	The text of the article(s) the essays were written in response to.

Table 1: Summary of the structure of the train_prompts.csv file

In our work we decided to *ignore* this file because we didn't use any of the

information contained in it, referring to the training part. More information about the inference and data augmentation part will be given later on.
train_essay.csv: Is a file that contains the essays written by the students and LLMs. This file is the *main* file of the dataset, in fact it contains the following fields:

Field	Description
id	A unique identifier for each essay.
prompt_id	Identifies the prompt the essay was written in response to.
text	The essay text itself.
generated	Whether the essay was written by a student (0) or generated by an LLM (1).

Table 2: Summary of the structure of the test—train_essays.csv file

2.1.1 Data Augmentation

The **main problem** of this competition is that the dataset is **incredibly unbalanced**, having less than the 1% of the data generated by LLMs.

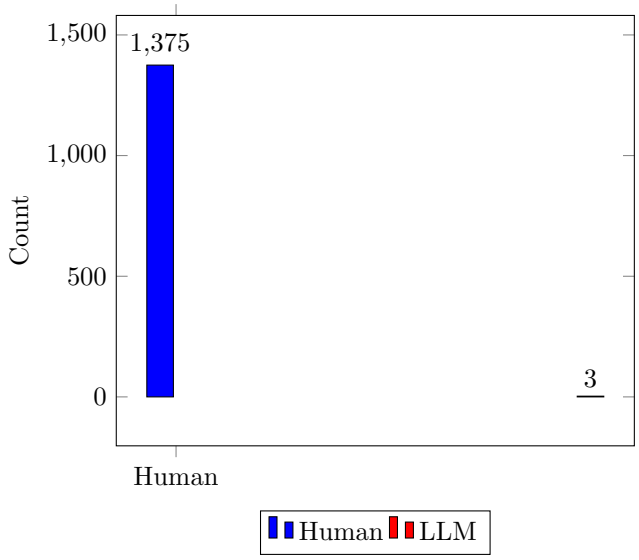


Figure 1: Distribution of the data

This is clearly a problem because the model will be biased towards the human essays, performing bad in detecting the LLMs. Other than that, 1375 might be a low number of essays to train the model and make it able to generalize.
Thus, it is needed to **augment** the dataset with **generated essays**. We decided to give a try to this approach, and used Mistral, a large language model of 7B

parameters. However, this approach has a big issue, computational power. In fact, to generate 20 essays using a free Kaggle account with GPU acceleration takes about 20 minutes. This means that to generate 1000 essays we would need around 17 hours.TPUs were not an option, since the time usage is limited and we would require it to fit the models.Due to that,we checked the *discussion* section on the competition, in which the community produced a variety of datasets. Our proposed models are using a combination of the following datasets:

- Daigt-v2
- LLM - Detect AI Generated Text - deobfuscation
- Daigt-v3

At the end, we managed to have a balanced dataset, so to avoid any kind of bias coming from unbalanced classes.
Code used for the test on data augmentation can be found in the provided notebooks.

2.2 Preprocessing

Before building a model to solve the problem we had to process the data in a way that it could be used in the training phase.
Our preprocessing phase mainly consists of removing punctuation, newline characters, double spaces. Then, since text cannot be fed to neural networks as it is, we had to **tokenize** them. More informations about this process can be found in the notebooks.

2.3 Development methodology and tools

To construct our model, we used mainly Tensorflow and Keras, with a few tests that involved PyTorch. Since it was not computationally possible to train on our computers, we used Kaggle notebooks to do that.
To speed up training, which was in some cases really time-consuming, we used the free quota of TPU offered by Kaggle, with an average reduction in times of $\sim$80%.

3 Models

3.1 Feedforward neural network

The first, basic approach that we tried was to tokenize each sentence in our dataset and then feed the token ids to a simple feedforward neural network. The model has the following structure:

Layer (type)	Output Shape	Param #
embedding_4 (Embedding)	(None, 512, 256)	2,560,000
flatten_4 (Flatten)	(None, 131,072)	0
dense_12 (Dense)	(None, 512)	67,109,376
dropout_9 (Dropout)	(None, 512)	0
dense_13 (Dense)	(None, 256)	131,328
dropout_10 (Dropout)	(None, 256)	0
dense_14 (Dense)	(None, 128)	32,896
dropout_11 (Dropout)	(None, 128)	0
dense_15 (Dense)	(None, 1)	129

with the following size:

Total params	69,833,729 (272.78 MB)
Trainable params	69,833,729 (272.78 MB)
Non-trainable params	0 (0.00 Byte)

3.1.1 Results

Even if the model converged to a 0.99 validation accuracy in a few epochs, the submission score was ∼0.6, so we concluded that this path was not really an optimal one. Then, we moved to different approaches already from an early stage.

3.2 Bert classifier

A simple but yet effective architecture that we build is a fine-tuned version of Bert, a pretrained model using a transformer architecture that can be used in a variety of text-based contexts with low fine-tuning[1].
The produced model has the following shape:

Layer (type)	Output Shape	Param #	Connected to
input_ids (InputLayer)	(None, None)	0	[]
attention_masks (InputLayer)	(None, None)	0	[]
tf_bert_model (TFBertModel)	[...]	1,094,822	{'input_ids[0][0]', 'attention_masks[0][0]'}
avgPool1 (GlobalAveragePooling1D)	(None, 768)	0	{'tf_bert_model[0][0]'}
dense (Dense)	(None, 512)	393,728	{'global_average_pooling1d[0][0]'}

with size:

Total params	109,876,481 (419.15 MB)
Trainable params	109,876,481 (419.15 MB)
Non-trainable params	0 (0.00 Byte)

3.2.1 Results

Wih a training phase of ten epochs with early-stopping criterion, we achieved a validation accuracy of ∼0.99 and a top score in the competition of 0.86.

3.3 Contrastive neural network

To test out some more complex architecture that could better fit our use case, we expanded the contrastive learning approach.
The main concept is around the training of an encoder, which is a model that can reduce the initial features to a smaller set of features in the **latent space**. The goal of this phase is to have the encoder to represent data in the same class close to each other, in a sort of labelled cluster.
Having this done, the representation of the data given in output by the encoder with freezed weights can be fed to a feedforward neural network to create separation surfaces inside the latent space to classify new points.

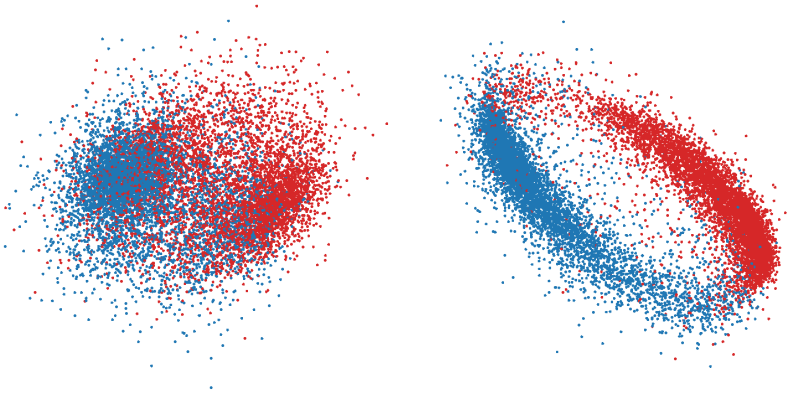


Figure 2: Embedding space of the encoder pre and post training phase.

Resulting models have the following shapes:

Layer (type)	Output Shape	Param #	Connected to
input_ids (InputLayer)	(None, 512)	0	[ ]
attention_masks (InputLayer)	(None, 512)	0	[ ]
text-encoder (Functional)	(None, 768)	1094822	{'input_ids[0][0]', 'attention_masks[0][0]'}

Layer (type)	Output Shape	Param #	Connected to
input_ids (InputLayer)	(None, 512)	0	[ ]
attention_masks (InputLayer)	(None, 512)	0	[ ]
text-encoder (Functional)	(None, 768)	1,094,822	{'input_ids[0][0]', 'attention_masks[0][0]'}
dropout_74 (Dropout)	(None, 768)	0	{'text-encoder[1][0]'}
dense_1 (Dense)	(None, 512)	393,728	{'dropout_74[0][0]'}
dropout_75 (Dropout)	(None, 512)	0	{'dense_1[0][0]'}
dense_2 (Dense)	(None, 1)	513	{'dropout_75[0][0]'}

and the following sizes:

Trainable params	590,592 (2.25 MB)
Non-trainable params	109,482,240 (417.64 MB)

Total params	109,876,481 (419.15 MB)
Trainable params	394,241 (1.50 MB)
Non-trainable params	109,482,240 (417.64 MB)

3.3.1 Results

In a few epochs, around ten, we could get a validation accuracy of ∼0.99, with the top score that we could get from the Kaggle submission is 0.76.

3.4 CNN architectures

3.4.1 Main concept

The main idea of CNN architectures is to apply the CNN methodology over the matrix of words embeddings of each sentence. To reach this, we first tokenized the whole dataset using the DistilBert tokenizer from Transformers library, then we applied the keras.layers.Embeddings layer.
Main reference in literature was the paper *Sentiment classification using convolutional neural networks.*[2]

3.4.2 CNN

After consulting the literature on the topic, the first test over a CNN lead us to create the following structure.

Layer (type)	Output Shape	Param #
tokenized_dataset (InputLayer)	[(None, 512)]	0
embeddings (Embedding)	(None, 512, 256)	2097152
reshape (Reshape)	(None, 512, 256, 1)	0
convolution-1 (Conv2D)	(None, 510, 254, 64)	640
convolution-2 (Conv2D)	(None, 508, 252, 32)	18464
maxpool1 (MaxPooling2D)	(None, 254, 126, 32)	0
flatten (Flatten)	(None, 1024128)	0
dense (Dense)	(None, 128)	131088512
dense_1 (Dense)	(None, 64)	8256
dense_2 (Dense)	(None, 16)	1040
dense_3 (Dense)	(None, 1)	17

The output model, with validation accuracy ∼0.99, has the following size.

Total params	133,214,081 (508.17 MB)
Trainable params	133,214,081 (508.17 MB)
Non-trainable params	0 (0.00 Byte)

3.4.3 *small* CNN

The first model, even if very precise, was a bit too large for us(∼ 512Mb). In the process to reduce its size while keeping the evaluation metrics constant, we ended up with the following schema.

Layer (type)	Output Shape	Param #
tokenized_dataset (InputLayer)	[(None, 512)]	0
embeddings (Embedding)	(None, 512, 128)	1,048,576
reshape (Reshape)	(None, 512, 128, 1)	0
convolution-1 (Conv2D)	(None, 510, 126, 64)	640
dropout (Dropout)	(None, 510, 126, 64)	0
convolution-2 (Conv2D)	(None, 508, 124, 32)	18,464
maxpool1 (MaxPooling2D)	(None, 254, 62, 32)	0
flatten (Flatten)	(None, 503,936)	0
dense (Dense)	(None, 64)	32,251,968
dense_1 (Dense)	(None, 16)	1,040

That, while keeping the metrics basically unchanged, resulted in a smaller model with dimensions as follows.

Total params	33320705 (127.11 MB)
Trainable params	33320705 (127.11 MB)
Non-trainable params	0 (0.00 Byte)

3.4.4 *big* CNN

The other two models were performing very good on our dataset, with great results on the submission to the competition as well (see Results section). But,

since we wanted to further increase our competition score, we developed a bigger version, with more feature extraction and less overfitting to the dataset.

Layer (type)	Output Shape	Param #
tokenized_dataset (InputLayer)	[(None, 512)]	0
embeddings (Embedding)	(None, 512, 256)	2,097,152
reshape (Reshape)	(None, 512, 256, 1)	0
convolution-1 (Conv2D)	(None, 510, 254, 64)	640
dropout (Dropout)	(None, 510, 254, 64)	0
convolution-2 (Conv2D)	(None, 508, 252, 32)	18,464
maxpool1 (MaxPooling2D)	(None, 254, 126, 32)	0
flatten (Flatten)	(None, 1,024,128)	0
dense (Dense) with regularizers(l1,l2)	(None, 256)	262,177,024
dense_1 (Dense)	(None, 64)	16,448
dense_2 (Dense)	(None, 16)	1,040
dense_3 (Dense)	(None, 1)	17

The final size of our big CNN is the following.
However, this did not cause a significant improvement on our leaderboard score.

Total params	264,310,785 (1008.27 MB)
Trainable params	264,310,785 (1008.27 MB)
Non-trainable params	0 (0.00 Byte)

3conv CNN

As we could test, changing the number of kernels in the convolutional stage or the number of neurons in the dense one was not a working path. We then ended up revising our strategy, moving it towards *architectural* changes instead of *parametric* ones.
Thinking of the domain, one of the problems could be overfitting on the train test: in fact, catching patterns that were related to the dataset over those that were naturally in the different statistic distribution of the two classes could lead in a bias and consequentially in a misclassification behaviour.
A solution to this problem could be the reduction of extracted features, but instead of reducing them in a parametric way we ended up *adding extra convolutional layers*.

Layer (type)	Output Shape	Param #
tokenized_dataset (InputLayer)	[(None, 512)]	0
embeddings (Embedding)	(None, 512, 256)	2,097,152
reshape_1 (Reshape)	(None, 512, 256, 1)	0
convolution-1 (Conv2D)	(None, 510, 254, 128)	1,280
dropout_2 (Dropout)	(None, 510, 254, 128)	0
convolution-2 (Conv2D)	(None, 508, 252, 64)	73,792
maxpool1 (MaxPooling2D)	(None, 254, 126, 64)	0
convolution-3 (Conv2D)	(None, 252, 124, 32)	18,464
maxpool2 (MaxPooling2D)	(None, 126, 62, 32)	0
maxpool3 (MaxPooling2D)	(None, 63, 31, 32)	0
flatten_1 (Flatten)	(None, 62,496)	0
dense_4 (Dense)	(None, 128)	7,999,616
dropout_3 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 64)	8,256
dense_6 (Dense)	(None, 16)	1,040
dense_7 (Dense)	(None, 1)	17

This caused a significant improvement in our scores, as it can be seen in the Results section. The resulting model has a significant reduction also in size,being the smallest cnn that we produced.

Total params	10,199,617 (38.91 MB)
Trainable params	10,199,617 (38.91 MB)
Non-trainable params	0 (0.00 Byte)

3.4.5 results

We noticed that this architecture converges really fast to a very high binary accuracy. Overall, the model was always performing really well (accuracy > 0.9) in few iterations of the very first epoch, being constant after at most 2-3 epochs.

Modello	**Validation Accuracy**	**Kaggle Score**	**Parameters Size**
cnn	0.99	0.804	508.17MB
small cnn	0.99	0.806	127.10MB
big cnn	0.99	0.802	1008.27MB
3conv cnn	0.99	0.852	38.91MB

4 Conclusions

4.1 Final Results

The final models produced scored in a very good way in proportion to our knowledge and computational resources. We also discovered new techniques to solve the problem object of the competition (and even other architectures not

part of the project).
We also feel more confident using deep learning libraries and environments.

4.2 Proof of Challenge Participation

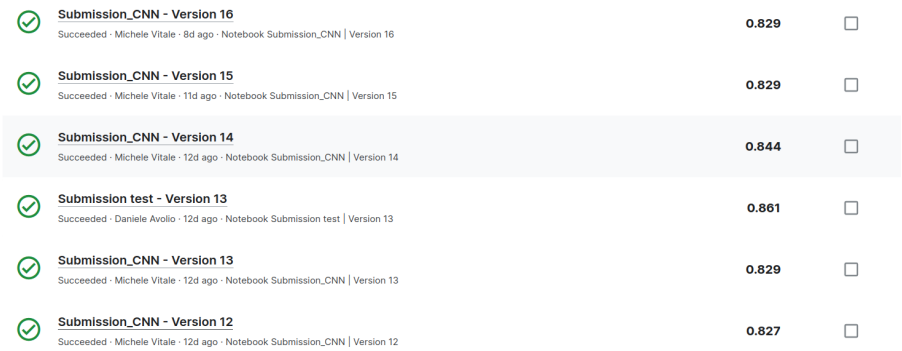


Figure 3: Some of our submissions

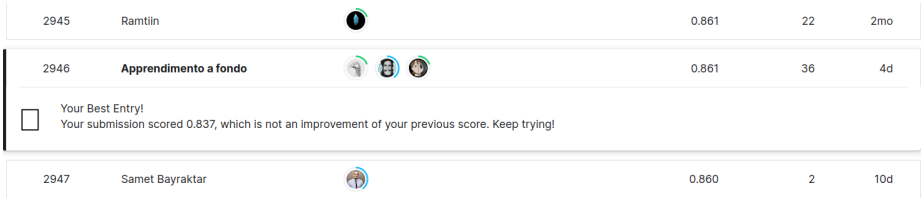


Figure 4: Leaderboard [16-01-24]

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[2] Hannah Kim and Young-Seob Jeong. Sentiment classification using convolutional neural networks. *Applied Sciences*, 9(11):2347, 2019.

A Appendix

This section contains all the plots and images that are coming out of our work. They consist of a different section because they are **not strictly required** to fully understand the report and they are also inside **related notebooks**. However, for completeness, they have been put here.

A.1 Feedforward Neural Network

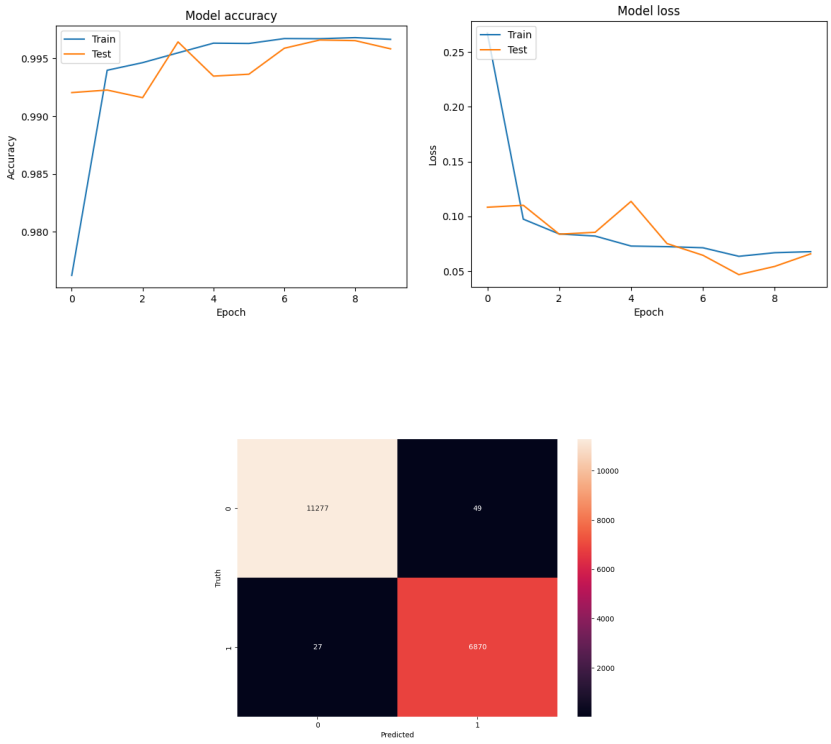


Figure 5: Feedforward neural network metrics (10 epochs)

A.2 Finetuned BERT

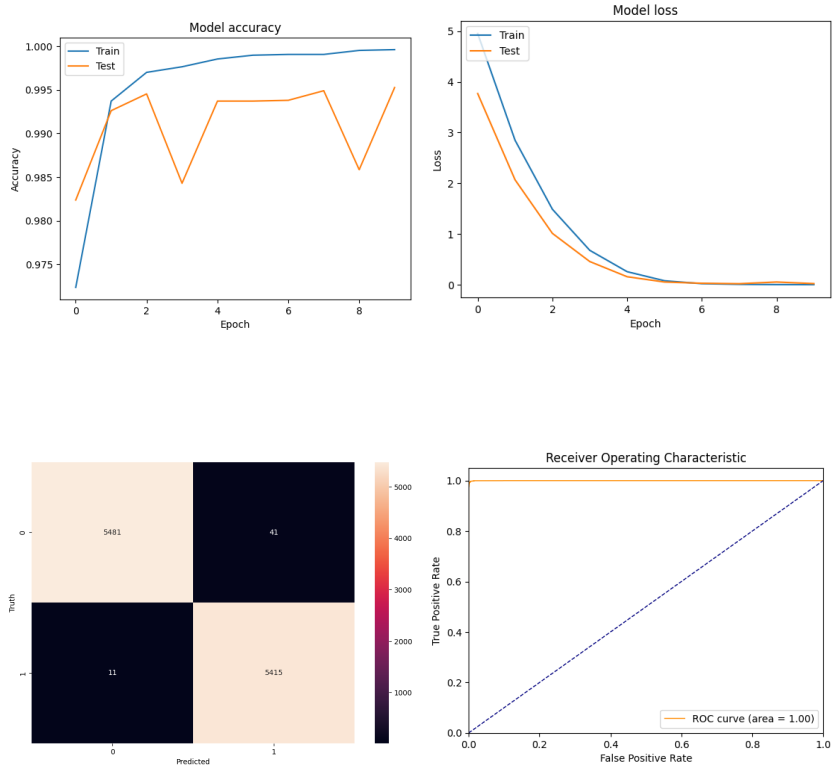


Figure 6: Finetuned BERT (10 epochs, with early-stopping)

A.3 Contrastive Neural Network with BERT

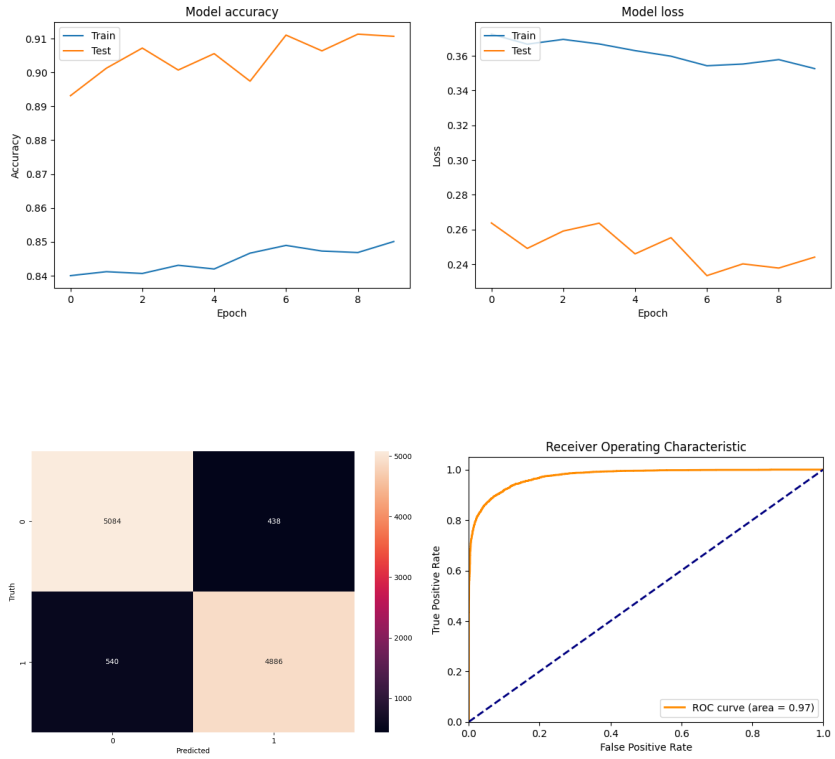


Figure 7: Contrastive neural network (10 epochs, with early-stopping)

A.4 Convolutional Neural Network

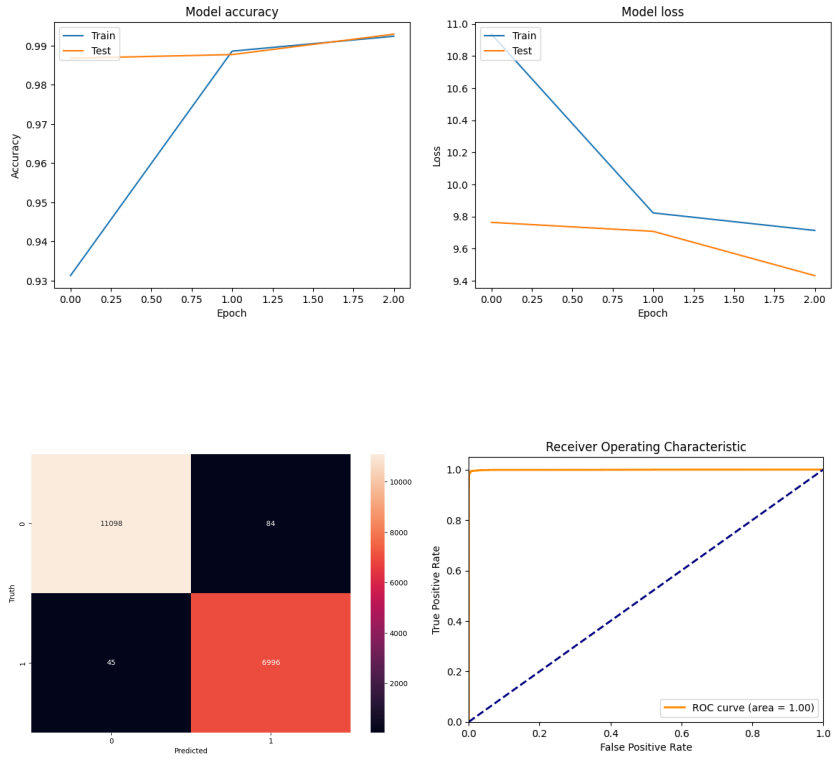


Figure 8: Convolutional neural network (3 epochs, *3conv* architecture)