Politecnico di Torino

Bioinformatics

Project 9
Diabetic Retinopathy scoring using GANs

Abbamonte Matteo 277483
Koudounas Alkis 278266

Referent: Prof. Alessio Mascolini

November 8, 2021

1 Project Focus

The main target of the project was that of using a GAN (Generative Adversarial Network) to perform semi-supervised scoring of Diabetic Retinopathy from images. In particular, as described in the specifics of the project, these steps have been followed:

- First, the Kaggle's Diabetic Retinopathy Detection dataset was downloaded and used. In particular, we managed to operate with the Btgraham-300 version of the dataset.
- Second, two different GAN architectures were proposed and trained with Google Colab for 400 epochs each, for a total amount of almost 600 hours training.
- Third, relevant features were extracted from the GAN discriminator, by building a model, namely *Extractor*, obtained by taking its layers until the Flatten and Fully Connected ones. A *Global Average pooling* layer was further used at the end to reduce the number of extracted features and allow us to work with the RAM offered by Google Colab.
- Fourth, a linear regressor to predict the clinical score from these extracted features was trained. Actually, we propose several alternatives, including *SKLearn Linear Regressor*, a model based on a Keras Dense layer, and another optimization process, based on the analytical coefficients of a linear function, namely M and Q.
- Finally, the performance of such regression algorithms were evaluated, and the results are reported in Subsection 2.5.

Section 2 reports the project implementation details, while Section 3 is a manual containing the sequence of instructions to perform in order to reproduce the experiments.

2 Implementation Details

In the following section, the five steps reported before are described in detail. In particular, Subsection 2.1 focuses on the characteristics of the dataset we were provided with, Subsection 2.2 outlines the architectures of the proposed models in all their versions, Subsection 2.3 explains how the task of features extractions is performed. Lastly, Subsection 2.4 highlights the algorithm proposed to implement the linear regressors, while Subsection 2.5 focuses on the results of the conducted experiments.

2.1 Dataset

The Kaggle's Diabetic Retinopathy Detection dataset is a large set (83.23 GB) of high-resolution retina images taken under a variety of imaging conditions. A left and right field is provided for every subject, and images are labeled with a subject ID, as well as either left or right. The images in the dataset come from different models and types of cameras, which can affect the visual appearance of left sample vs. right sample. Like any real-world data set, there is noise in both the images and labels: images may contain artifacts, be out of focus, underexposed, or overexposed.
A clinician has rated the presence of diabetic retinopathy in each image on a scale of 0 to 4, according to the following scale:

- 0 - No DR
- 1 - Mild
- 2 - Moderate
- 3 - Severe

- 4 - Proliferative DR

In particular, we used the Btgraham-300 version of this dataset. Benjamin Graham (the winner of the Kaggle competition related to the publication of the dataset) proposed indeed a solution to pre-process these images in order to remove some of the variation between the images due to differing lighting conditions, camera resolution, etc. These are the steps he performed (more details can be found in this file[1]):

1. Rescaling the images to have the same radius, that is 300 pixels or 500 pixels (we used the version with 300 px)
2. Subtracting the local average color; the local average gets mapped to 50% gray
3. Clipping the images to 90% size to remove the "boundary effects"

Fig. 1 reports two before/after examples, taken from the above linked file.

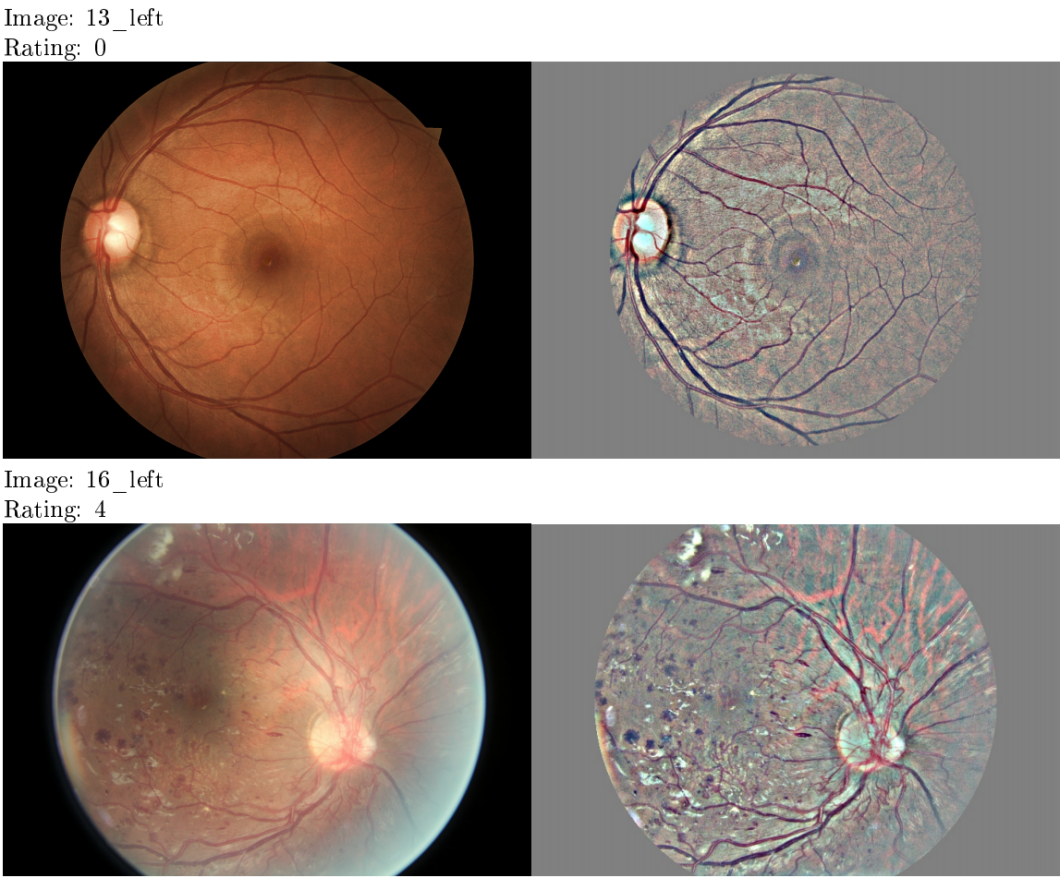


Figure 1: Two images from the training set. Original images on the left and Btgraham-300 pre-processed version on the right.

2.2 GAN architectures

To achieve the goal of the project, two different DC-GAN architectures are proposed in the following. As it is possible to notice, for each GAN three steps are performed, with increasing network depth.

[1]Link to the Btgraham-300 report file.

```
Model: "discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 32)      1568
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 256, 256, 32)      0
_________________________________________________________________
dropout (Dropout)            (None, 256, 256, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 64)      32832
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 128, 128, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128, 128, 64)      0
_________________________________________________________________
flatten (Flatten)            (None, 1048576)           0
_________________________________________________________________
dense (Dense)                (None, 1)                 1048577
=================================================================
Total params: 1,082,977
Trainable params: 1,082,977
Non-trainable params: 0
_________________________________________________________________
```

(a) Discriminator of the first GAN, first step

```
Model: "generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 2097152)           209715200
_________________________________________________________________
batch_normalization (BatchNo (None, 2097152)           8388608
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 2097152)           0
_________________________________________________________________
reshape (Reshape)            (None, 128, 128, 128)     0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 128, 128, 64)      131072
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 128, 64)      256
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 256, 256, 32)      32768
_________________________________________________________________
batch_normalization_2 (Batch (None, 256, 256, 32)      128
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 256, 256, 32)      0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 512, 512, 3)       1536
=================================================================
Total params: 218,269,568
Trainable params: 214,075,072
Non-trainable params: 4,194,496
_________________________________________________________________
```

(b) Generator of the first GAN, first step

Figure 2: First GAN Architecture, first step

```
Model: "discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 32)      1568
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 256, 256, 32)      0
_________________________________________________________________
dropout (Dropout)            (None, 256, 256, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 64)      32832
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 128, 128, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128, 128, 64)      0
_________________________________________________________________
flatten (Flatten)            (None, 1048576)           0
_________________________________________________________________
dense (Dense)                (None, 1)                 1048577
=================================================================
Total params: 1,082,977
Trainable params: 1,082,977
Non-trainable params: 0
_________________________________________________________________
```

(a) Discriminator of the second GAN, first step

```
Model: "generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 2097152)           209715200
_________________________________________________________________
batch_normalization (BatchNo (None, 2097152)           8388608
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 2097152)           0
_________________________________________________________________
reshape (Reshape)            (None, 128, 128, 128)     0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 256, 256, 64)      131072
_________________________________________________________________
batch_normalization_1 (Batch (None, 256, 256, 64)      256
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 256, 256, 64)      0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 512, 512, 32)      32768
_________________________________________________________________
batch_normalization_2 (Batch (None, 512, 512, 32)      128
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 512, 512, 32)      0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 512, 512, 3)       1536
=================================================================
Total params: 218,269,568
Trainable params: 214,075,072
Non-trainable params: 4,194,496
_________________________________________________________________
```

(b) Generator of the second GAN, first step

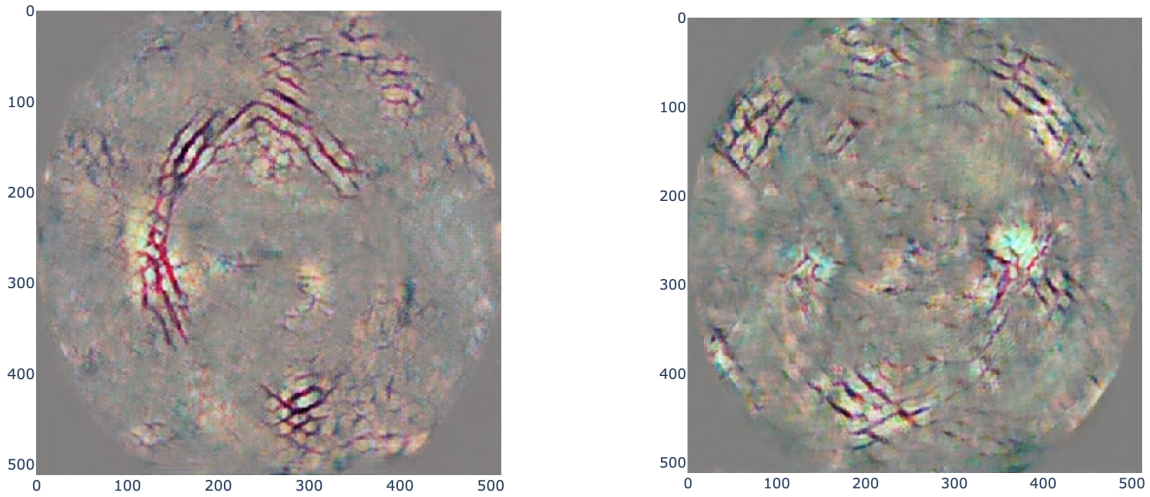
Figure 3: Second GAN Architecture, first step

2.2.1 First Step

First GAN. The generator of the first GAN, in its first version, uses three transposed convolution layers for upsampling (as seen in different implementations), three batch normalization layers, an initial fully-connected layer and Leaky ReLU as activation function. The discriminator of the first GAN instead, in its first version, uses two 2D convolutional layers for downsampling, with dropout and leaky ReLU as activation function. Fig. 2 shows the overall architecture of the model.

Second GAN. The generator of the second GAN, in its first version, uses the same structure as the first GAN, but the dimensions of the generated images are incremented early in the pipeline by using a bigger stride in the first transposed convolutional layer. In this way, the features are earlier upsampled too. The discriminator of the second GAN, in its first version, is exactly the same as the one of the first GAN, as it is possible to see by looking at Fig. 3, that depicts the architecture of the model.

Results. Fig. 4 shows the images generated by the first and the second GANs. The training counts 200 epochs for each model, for a total amount of 89 hours of training for the first GAN and 122 hours for the second.

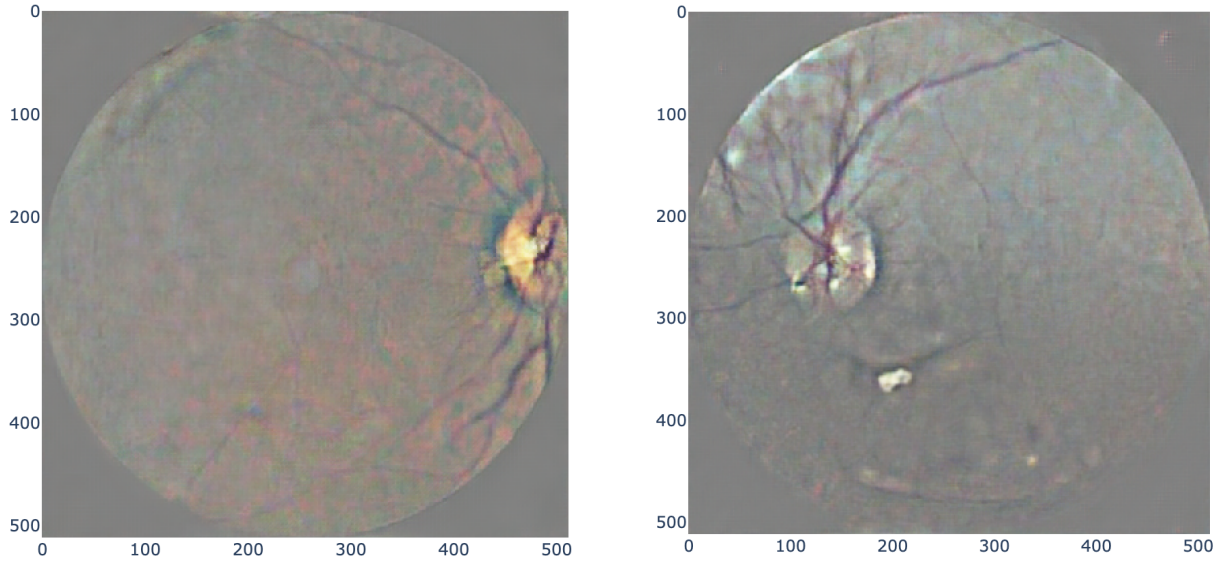


(a) Image generated by the first GAN, first step (b) Image generated by the second GAN, first step

Figure 4: Images generated by the two GANs, first step

2.2.2 Second Step

Since the number of features extracted was too high to be feasible for the linear regression, and the networks weren't trained enough to identify significant features, we thought about adding two more convolutional layers in the discriminator. Doing so, the total amount of extracted features at the end of the pipeline decreased of about one order of magnitude, and the performance of the discriminator increased significantly. In this process, we used the weights of the previous training checkpoints as an initialization for the pre-existing layers. In this way, we managed not to waste too much time for the training itself, since it took 50 more hours to reach an overall amount of 300 epochs for the first GAN, and 75 more hours to reach the same number of epochs of the second GAN.
Fig. 5 shows the results of the two GANs after this second training step.



(a) Image generated by the first GAN, second step (b) Image generated by the second GAN, second step

Figure 5: Images generated by the two GANs, second step

```
Model: "discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 32)      1568
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 256, 256, 32)      0
_________________________________________________________________
dropout (Dropout)            (None, 256, 256, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 64)      32832
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 128, 128, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 128)       131200
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 64, 64, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 256)       524544
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 32, 32, 256)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 512)       2097664
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 16, 16, 512)       0
_________________________________________________________________
dropout_4 (Dropout)          (None, 16, 16, 512)       0
_________________________________________________________________
flatten (Flatten)            (None, 131072)            0
_________________________________________________________________
dense (Dense)                (None, 1)                 131073
=================================================================
Total params: 2,918,881
Trainable params: 2,918,881
Non-trainable params: 0
_________________________________________________________________
```

(a) Discriminator of the first GAN, third step

```
Model: "generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 2097152)           209715200
_________________________________________________________________
batch_normalization (BatchNo (None, 2097152)           8388608
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 2097152)           0
_________________________________________________________________
reshape (Reshape)            (None, 128, 128, 128)     0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 128, 128, 64)      131072
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 128, 64)      256
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 256, 256, 32)      32768
_________________________________________________________________
batch_normalization_2 (Batch (None, 256, 256, 32)      128
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 256, 256, 32)      0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 512, 512, 3)       1536
=================================================================
Total params: 218,269,568
Trainable params: 214,075,072
Non-trainable params: 4,194,496
_________________________________________________________________
```

(b) Generator of the first GAN, third step

Figure 6: First GAN Architecture, third step

```
Model: "discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 32)      1568
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 256, 256, 32)      0
_________________________________________________________________
dropout (Dropout)            (None, 256, 256, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 64)      32832
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 128, 128, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 128)       131200
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 64, 64, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 256)       524544
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 32, 32, 256)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 512)       2097664
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 16, 16, 512)       0
_________________________________________________________________
dropout_4 (Dropout)          (None, 16, 16, 512)       0
_________________________________________________________________
flatten (Flatten)            (None, 131072)            0
_________________________________________________________________
dense (Dense)                (None, 1)                 131073
=================================================================
Total params: 2,918,881
Trainable params: 2,918,881
Non-trainable params: 0
_________________________________________________________________
```

(a) Discriminator of the second GAN, third step

```
Model: "generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 2097152)           209715200
_________________________________________________________________
batch_normalization (BatchNo (None, 2097152)           8388608
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 2097152)           0
_________________________________________________________________
reshape (Reshape)            (None, 128, 128, 128)     0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 256, 256, 64)      131072
_________________________________________________________________
batch_normalization_1 (Batch (None, 256, 256, 64)      256
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 256, 256, 64)      0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 512, 512, 32)      32768
_________________________________________________________________
batch_normalization_2 (Batch (None, 512, 512, 32)      128
_________________________________________________________________
leaky_re_lu_7 (LeakyReLU)    (None, 512, 512, 32)      0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 512, 512, 3)       1536
=================================================================
Total params: 218,269,568
Trainable params: 214,075,072
Non-trainable params: 4,194,496
_________________________________________________________________
```

(b) Generator of the second GAN, third step

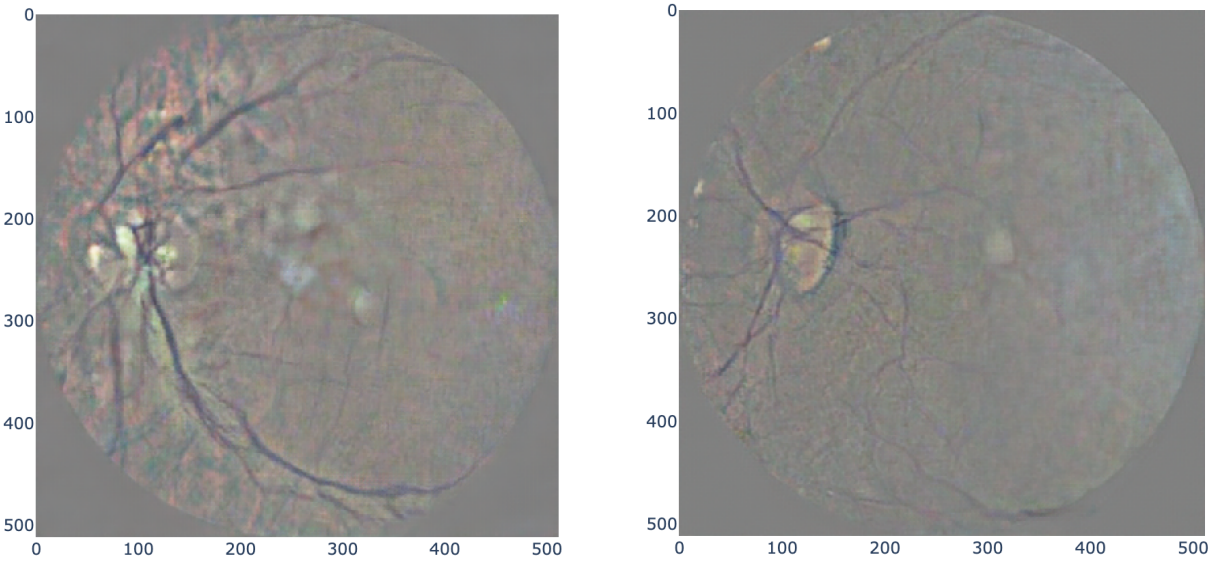
Figure 7: Second GAN Architecture, third step

2.2.3 Third Step

In order to further improve the performance of the discriminator and, accordingly, reduce the number of extracted features, we decided to modify again the discriminator itself, by adding another convolutional layer.
We settled to train the networks for the final 100 epochs, for a total amount of 261 hours. Again, we used the previous weights of the training checkpoints as an initialization for the current network.
Fig. 6 and Fig. 7 display the architecture of the models, respectively first and second GAN, while Fig. 8 shows the images produced by the two models.

2.3 Feature Extractor

In order to extract features by using the GAN Discriminator we used its layers until the fully-connected blocks, and we applied a Global Average pooling layer in order to reduce the number of features to a more appropriate order of magnitude, that is, 512. Thus, we applied this feature extractor to the original dataset to create the corresponding features dataset.
This process is followed for each of the previously defined GAN architectures, by considering the third step setup.



(a) Image generated by the first GAN, third step

(b) Image generated by the second GAN, third step

Figure 8: Images generated by the two GANs, third step

2.4 Linear Regressor

To predict the clinical score from the features just obtained, we built three different linear regression models, using SKLearn implementation, a model based on a Keras Dense layer, and a process focused on the optimization of the coefficients of a linear function.
Since the dataset structure is inherently unbalanced (due to the preponderance of samples labelled with "class" (score) 0), we decided to apply a class weighting strategy.

SKLearn Implementation. Regarding the first attempt, we used a model based on the Least-Squares method provided by the Scikit-Learn library, that is the standard approach for linear regression. This method shows to be extremely fast, since it performs a single-pass optimization, taking the entire dataset for a single minimization step.

Analytical Solution. Another approach that we decided to take is that of implementing an analytical solution optimizing the coefficients of a linear function. In the provided code, M stands for the coefficients vector, containing both the angular coefficient and the known term.
Since the input is a features vector, also the angular coefficient is defined as a vector.
The optimization is based on a weighted ℓ_2 loss.

Keras Dense Layer. The final technique is based on the usage of a single Dense Layer that takes as input the features extracted from the images and outputs the clinical score.

2.5 Performance Evaluation

In order to evaluate which regressor model performs better, we decided to perform a stratified k-fold cross validation, with $k = 5$ and 2000 training epochs (for the models based on the optimization process). Looking at the validation losses, it is possible to state that the analytical optimization can reach similar results as the Least Squares approach in a decent number of epochs, while being extremely slower due to the optimization algorithm itself. On the other side, the model based on the Keras Dense layer turns out to be even slower than the analytical one, also getting slightly worse outcomes.
Overall results are summarized in Table 1.

	Linear Regressor Models	**Stratified K-Fold Cross Validation**	
		Validation Loss	**Computation Time (s)**
1st GAN	**SKLearn**	**0.9889**	**1.0928**
	Analytical	1.0508	83.8348
	Dense Layer	1.0691	245.2631
2nd GAN	**SKLearn**	**0.9778**	**1.0769**
	Analytical	1.0037	74.8295
	Dense Layer	1.0212	271.0011

Table 1: **Stratified K-Fold Cross Validation.** Results of the Stratified K-Fold Cross Validation (with $K = 5$ and 2000 training epochs), in terms of validation loss and time needed for the computation, considering the proposed linear regression models applied on the features extracted by the two GAN architectures.

As it is possible to see by reading Table 1, no matter the GAN taken into account, SKLearn outperforms all the other approaches, achieving the best results both for what concerns the validation loss and the time required for the computation (that is measured in seconds). The linear regressor based on the analytical optimization reaches almost the same results as the one using the Keras dense layer, while being significantly faster. In order to perform a fair comparison, we decided to train the "analytical" linear regressor for a bigger number of epochs, 5× the number used for the cross-validation (thus, 10000 epochs). Doing so, we hoped that this training could cover the gap existing between this linear regressor and the one based on SKLearn. To evaluate the performance of the linear regressors we decided to adopt the two most used metrics in this field, namely the "Mean Absolute Error" (MAE) and the "Root Mean Squared Error" (RMSE).

As shown in Table 2, despite the training procedure consisting of 10000 epochs applied on the "analytical" linear regressor, this couldn't reach the same performance of the SKLearn one, even if both the MAE and the RMSE are quite close one another. Overall the second GAN allowed us to achieve the best results by using SKLearn as well as the Analytical solution, although the results are almost comparable. In order to have also a visual idea about the scores predicted starting from the features extracted by the two GANs, one could take a look on Fig. 9 for the first GAN and Fig. 10 for the second. While not being extremely accurate, a note has to be made. Before the usage of the class re-weighting strategy, all the predicted scores were biased to the 0, since that was the class with the

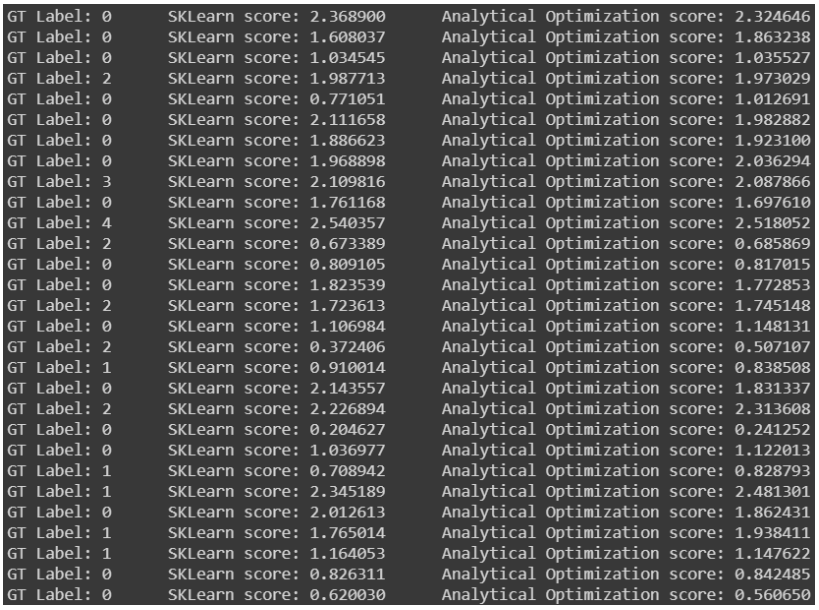


Figure 9: **Scores predictions based on the First GAN feature extractor**. Scores predicted by the SKLearn and the Analytical Optimization models (on the right) with respect to the ground truth labels (on the left), based on the features extracted by means of the first GAN.

Linear Regressor Models		Performance Evaluation	
		MAE	**RMSE**
1st GAN	**SKLearn**	**1.1217**	**1.3095**
	Analytical	1.1290	1.3155
2nd GAN	**SKLearn**	**1.1157**	**1.3005**
	Analytical	1.1172	1.3011

Table 2: **Performance Evaluation.** MAE and RMSE metrics for the two Linear Regression models implemented, divided by the GAN used for the extraction of the features dataset.

higher number of samples. Now instead, even if not precisely, the results are well-spread across the five different classes.

3 Experiments Manual

In this section there is a step-by-step explanation concerning the experiments exposed in the provided notebook[2]. In the following, the main (and the most easily reproducible) experiments are highlighted. Within the main explanations there are some operations that are not recommended (even if possible), due to their intensive need for time and power.

PRESETS section must always be run when starting the notebook. It contains two subsections, namely **Import Libraries** and **Utils**: the first imports the packages that are necessary for each experiment, while the second includes several functions (for the feature extraction phase) useful for summarizing processes already explained in the previous phases.
When running this cell, **it is important** to expand the "Import Libraries" Section and follow the Drive authentication procedure, by clicking on the provided link and completing the access and authentication process by using the following username and password: "generaldrive21@gmail.com", "bioinf2021".

DIABETIC RETINOPATHY DETECTION DATASET section **should not be run**, since it

[2]Link to the notebook.

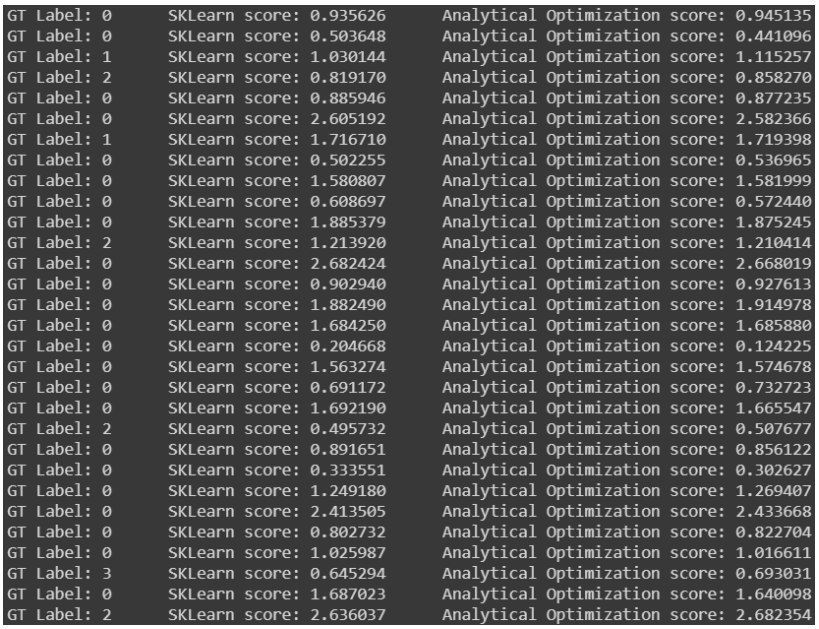


Figure 10: **Scores predictions based on the Second GAN feature extractor**. Scores predicted by the SKLearn and the Analytical Optimization models (on the right) with respect to the ground truth labels (on the left), based on the features extracted by means of the second GAN.

is the sequence of commands that allowed us to download the original dataset, decompress it, prepare it and build it, so that we could use it for subsequent attempts and experiments.

RETINOPATHY SAMPLE GENERATION section contains the two networks that are implemented, thus two main paths can be followed depending on which GAN one wants to test. Nonetheless, one could even choose to run the entire section to see the results provided by both the GANs.

1. **Load Dataset for GAN Training** subsection is used to build and batch the dataset that is pre-saved in the mounted *gdrive* partition. However it is useful only if one wants to train the network, and this is the reason why the provided code is commented.

2. **First GAN** and **Second GAN** subsections have the same structure. They can be run to build the model, restore the latest checkpoint for the corresponding GAN and finally obtain a new image, that will be plotted.
   The **Training** sub-subsection inside each GAN subsection is currently commented, since it starts the training process, starting from the latest checkpoint.

FEATURE EXTRACTION section is characterized by the same two-paths structure as the previous one, each loading the corresponding GAN model.
In order to properly run a feature extraction process, a choice between one of the two provided bases for the extractor should be made and the corresponding subsection should be run.
Each subsection includes a call to a function for restoring the chosen GAN and build the features extractor model starting from it, as well as the code for performing the features extraction itself.

CROSS-VALIDATION section starts the corresponding phase. It includes the features extraction phase (by means of the best performing GAN model for the extractor itself), and a stratified 5-fold cross-validation process based on the main linear regression models that have been analyzed, namely SKLearn, Analytical Optimization and Keras Dense Layer.

LINEAR REGRESSION section should be run entirely. It contains the code for building the two best Linear Regression models analyzed (the one based on SKLearn and the other that uses the

analytical optimization), a function for computing the weights for the loss based on the occurrences of the labels, a call to a function that restores the best GAN model for the feature extraction phase and the instructions for fitting or training the models, with the corresponding visualization of a batch of predictions and the metrics computation.