Università della Calabria

Department of Mathematics and Computer Science

Master Degree Course in Artificial Intelligence and Computer Science

Deep Lerning project report

Emanuele Conforti - 252122

Jacopo Garofalo - 252093

Gianmarco La Marca - 252256

Academic Year 2024/2025

Contents

Chapter 1

Introduction

In this project, we focus our work in performing a generative task using the Chest X-ray dataset from Indiana University (https://www.kaggle.com/datasets/raddar/chest-xrays-indiana-university), which contains a set of data related to some medical examinations that occurred in the United States, accompanied by chest X-ray images properly labeled and their corresponding medical reports.
The goal of the project is to generate a representation of a medical report starting with a chest X-ray image, ensuring that the X-ray aligns with the content of the report.
The project can be divided in two sub-tasks, each with a different sub-goal:

1. **Image reconstruction**: given a certain X-ray image, the model should be capable of accurately reconstructing it.

2. **Image-To-Text**: using the previously trained model for the images and an appropriate text decoder, the goal is to generate a medical report based on its corresponding image.

The combination of the models and results of each sub-task will be the pipeline of the main generative task of the project.

Chapter 2

Pre-Processing

Each record in the Chest X-ray dataset contains some information of the medical examination of a certain patient, including the patient ID, frontal chest X-ray images, lateral chest X-ray images, the corresponding findings (i.e., the medical report derived from the X-rays), and a set of tags that concisely describe the patient's pathologies (if any) as well as various elements present in the X-rays.

2.1 Data pre-processing

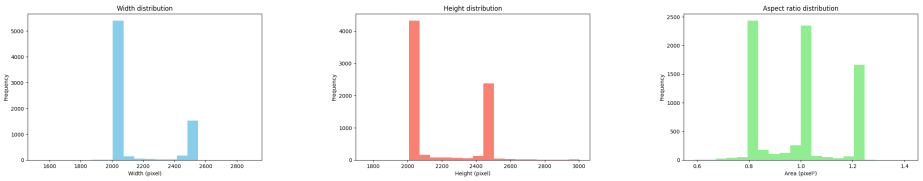
For our tasks, we need every entry in the dataset to be associated with a medical finding (useful for the text generation task) and at least one X-ray image (necessary for the image reconstruction task), so we removed all the entries that didn't respect all these conditions, noticing that majority of the removed entries were those without the medical finding.
The dataset was divided into training and validation sets following a 90%-10% ratio. We decided to use a sample of an external similar dataset (https://huggingface.co/datasets/Sina-Alinejad-2002/train_chexpert) as a test set due to the small size of the dataset, so the current ratio allow us to have a good amount of data on the training set. The split was performed on a patient basis, ensuring that both frontal and lateral X-rays of the same patient remained within the same subset (training or validation). This was done since splitting

on an X-ray basis could lead to an uneven distribution of frontal and lateral X-rays across subsets, potentially resulting in one subset containing only a single type of X-ray.

2.2 X-ray images pre-processing

Since the model requires input images of a uniform size to work better, it was necessary to standardize the dimensions of the chest X-ray images to a common width and height. To achieve this, there are three possible approaches:

- Resizing the images to a fixed size, which may introduce slight distortions, but it's the standard way to deal with images;
- Applying padding to maintain the original proportions;
- Cropping the images, which carries the risk of removing critical information.



(a) Width distribution (b) Height distribution (c) AR distribution

Figure 2.1: Image dimension distributions

We chose to resize the images to 224×224 for two main reasons:

- the variation in image sizes, as illustrated in Figure 2.1, leading us to find an average value of size, so the resizing would affect all the images in a uniform way.
- 224×224 are the standard dimensions for many pre-trained models, so that we can be already prepared for the eventual use of one of them.

Finally, given the lack of significant color information in the original X-Ray images, we reduced the channels from 3 to 1 to simplify and accelerate computations during training.

Chapter 3

The Model

In the following chapter, we will provide a detailed discussion of the model utilized to accomplish the project's objectives. Specifically, we will examine the architecture of each part of the model, represented by a sub-model, highlight its structural components and outline the training pipeline.

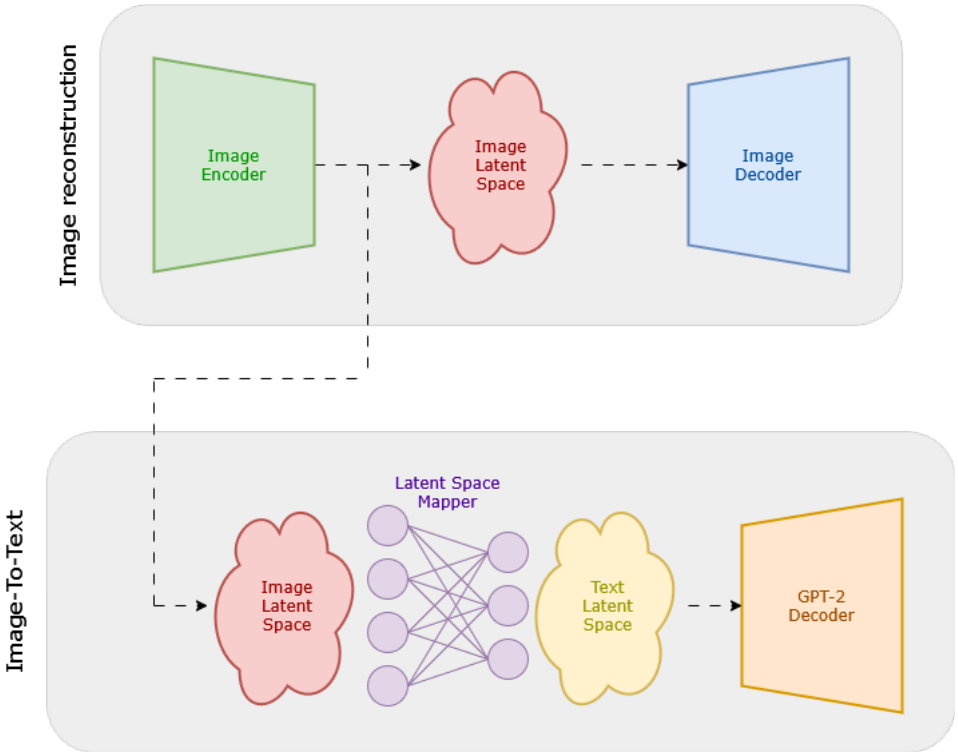


Figure 3.1: Model structure

Due to the interaction with different types of data and the generative nature of

the main task, the main model will have a multi-modal architecture, capable of combining models used for different tasks in a more complete one:

- **X-ray Image Encoder**: We train an encoder-decoder model, where an X-ray image is provided as input, and the model's purpose is to accurately reconstruct it. This process allows us to obtain a well-trained encoder, which will later be used in the main task;
- **Text Decoder**: We analyze different state-of-the-art models used for generating text, especially pre-trained transformers.
- **Latent Space Mapper**: We use a Feed-Forward Neural Network as a glue to map the latent space generated from the Image Encoder to the latent space requested from the Text Decoder.

In the following sections, we will provide a more in-depth analysis of each of these sub-models.

3.1 Image Encoder

The **Image Encoder** is the first module of our architecture, its purpose is to convert the input image into a latent space.
To do that, we constructed an **Autoencoder** model that reconstructs the input image, and then we used only the encoder in our architecture.

3.1.1 CNN Autoencoder

The CNN Autoencoder is composed of 2 modules:

- Encoder, that takes in input a 224x224 pixel image in gray-scale (only 1 channel) and gives in output 512 feature maps of dimention 12x12
- Decoder, that takes in input 512 feature maps of dimention 12x12 and reconstructs a single 224x224 pixel image in gray-scale (only 1 channel)

The python code for the implementation is the following:

```
class AutoEncoderCNN(nn.Module):

    def __init__(self, in_channels):
        super().__init__()
        self.encoder = nn.Sequential(
            conv_layer(in_channels, 64, 3),
            conv_layer(64, 128, 3),
            conv_layer(128, 256, 3),
            conv_layer(256, 512, 3)
        )
        # 12x12
        self.decoder = nn.Sequential(
            conv_transpose_layer(512, 256, 4),
            feature_recon_layer(256),
            conv_transpose_layer(256, 128, 4),
            feature_recon_layer(128),
            conv_transpose_layer(128, 64, 5),
            conv_transpose_layer(64, in_channels, 4)
        )

    def encode(self, x):
        return self.encoder(x)

    def decode(self, z):
        return self.decoder(z)

    def forward(self, x):
        encoded = self.encode(x)
        decoded = self.decode(encoded)

        return encoded, decoded
```

Several custom function where used in this implementation:

- **conv_layer**, which applies a convolution to the input using a specified dimension of the kernel.

  The code for this function is the following:

```
def conv_layer(n_input, n_output, kernel_size, stride=1):
    return nn.Sequential(
        nn.Conv2d(n_input, n_output, kernel_size, stride),
        nn.ReLU(),
        nn.BatchNorm2d(n_output),
        nn.MaxPool2d(2)
    )

```

  As can be seen, first a convolution is applied (with the kernel size specified in input) and this produces in output a number of feature maps equal to "n_output". Then, ReLU and Normalization layers are applied, and finally a MaxPool layer that halves the dimention of the feature maps.

- **conv_transpose_layer**, which applies the transpose convolution function to the input using a specified dimension of kernel.

  The code for this function is the following:

```
def conv_transpose_layer(n_input, n_output, kernel_size,
   stride=2):
    return nn.Sequential(
        nn.ConvTranspose2d(n_input, n_output, kernel_size,
    stride),
        nn.ReLU(),
        nn.BatchNorm2d(n_output)
    )

```

  As can be seen, first a transpose convolution is applied (with the kernel size specified in input and a stride of 2) and this produces in output a number of feature maps equal to "n_output". Then ReLU and Normalization layers are applied.

- **feature_recon_layer**, which applies a transpose convolution and then a subsequent convolution, that leaves the dimention of the input unchanged. This is done to avoid the loss of feature in the upscaling process of the decoder.

  The code for this function is the following:

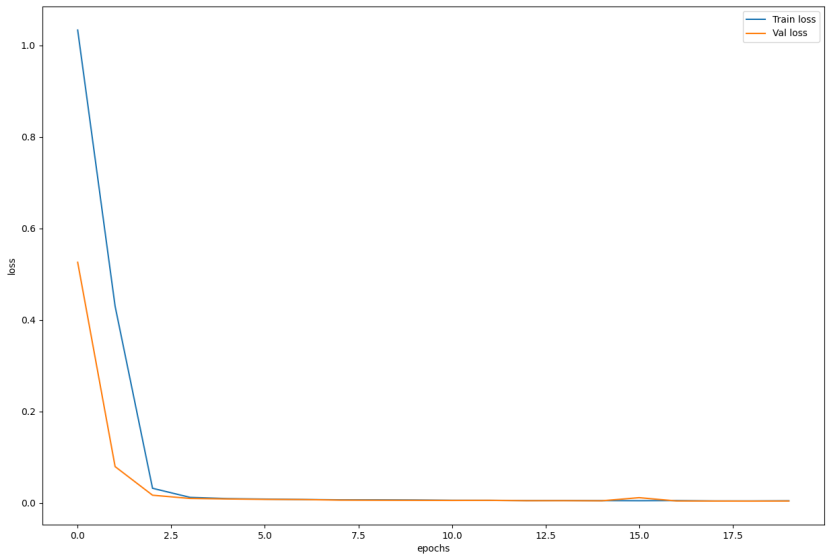
```
def feature_recon_layer(n_input):
    return nn.Sequential(
        conv_transpose_layer(n_input, n_input, 4, 1),     #
    dim += 3
        nn.Conv2d(n_input, n_input, 4, 1),                #
    dim -= 3
        nn.ReLU(),
        nn.BatchNorm2d(n_input)
    )

```

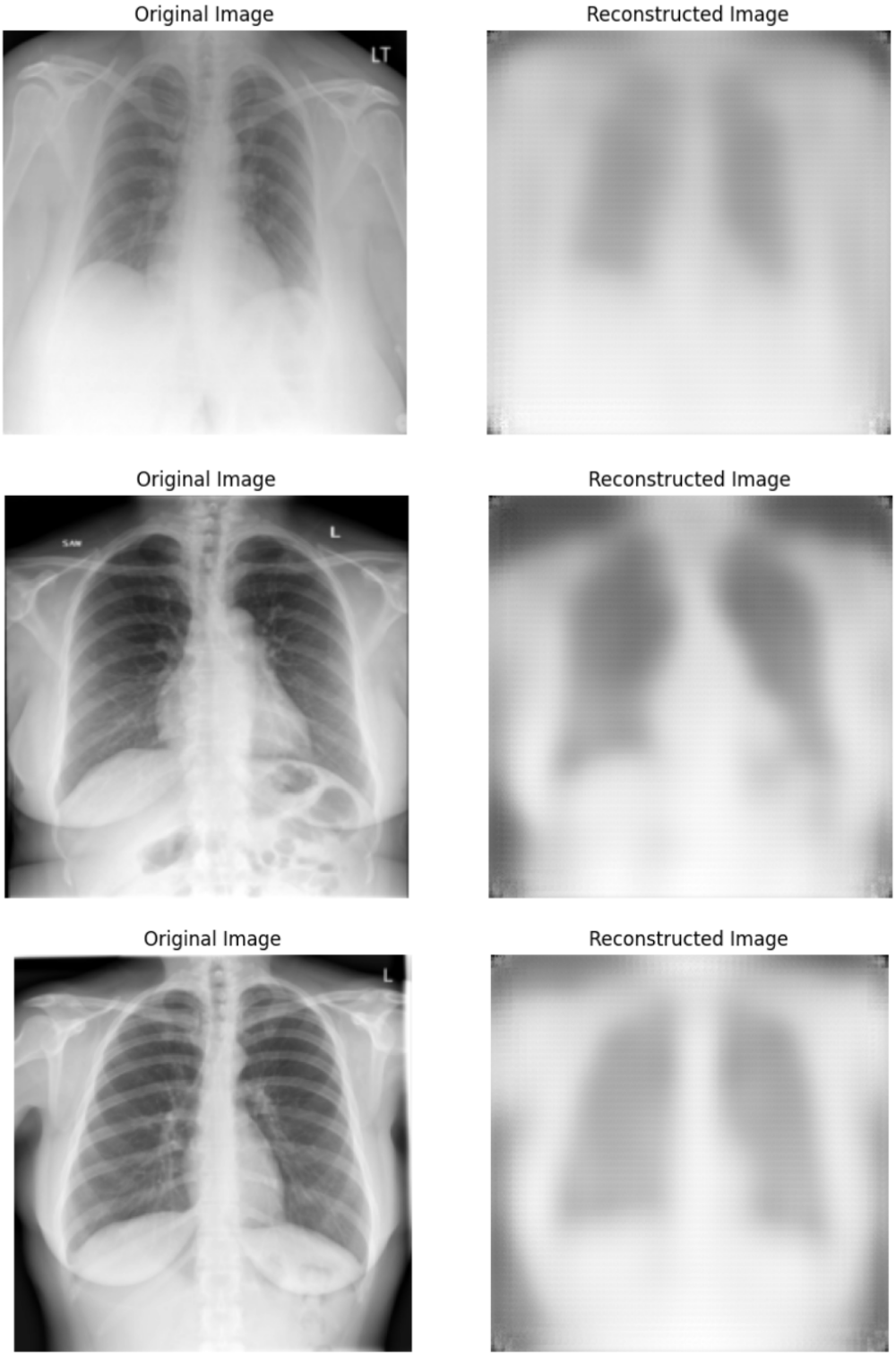
In our first attempts, to train the **Autoencoder** we used the MSE loss (Mean Squared Error) which presents the following function:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

The resulting loss function was the following:



The reconstructed images were the following:



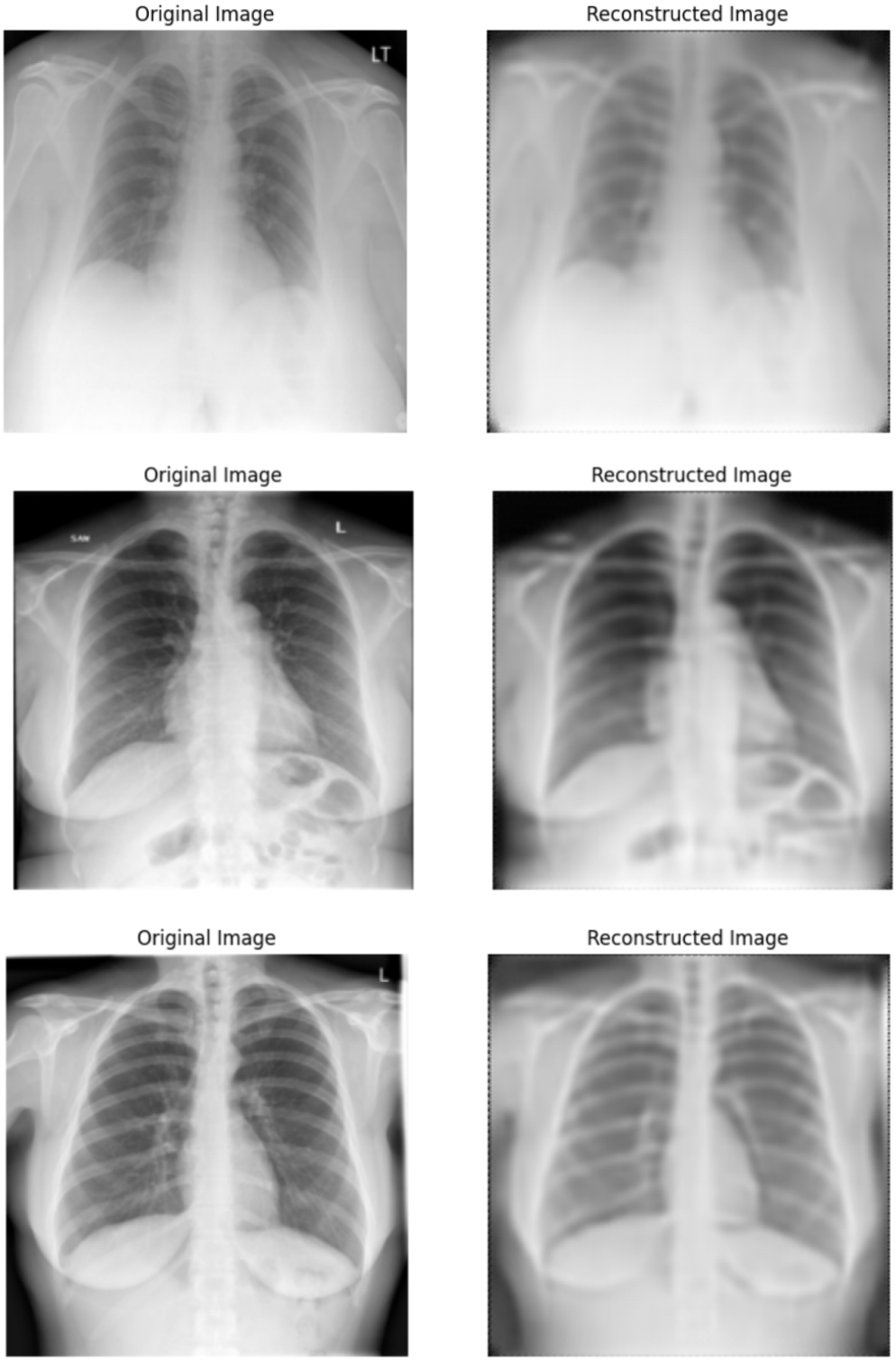
In the second variation we used as loss function the SSIM (Structural Similarity Index Measure), which presents the following formula:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

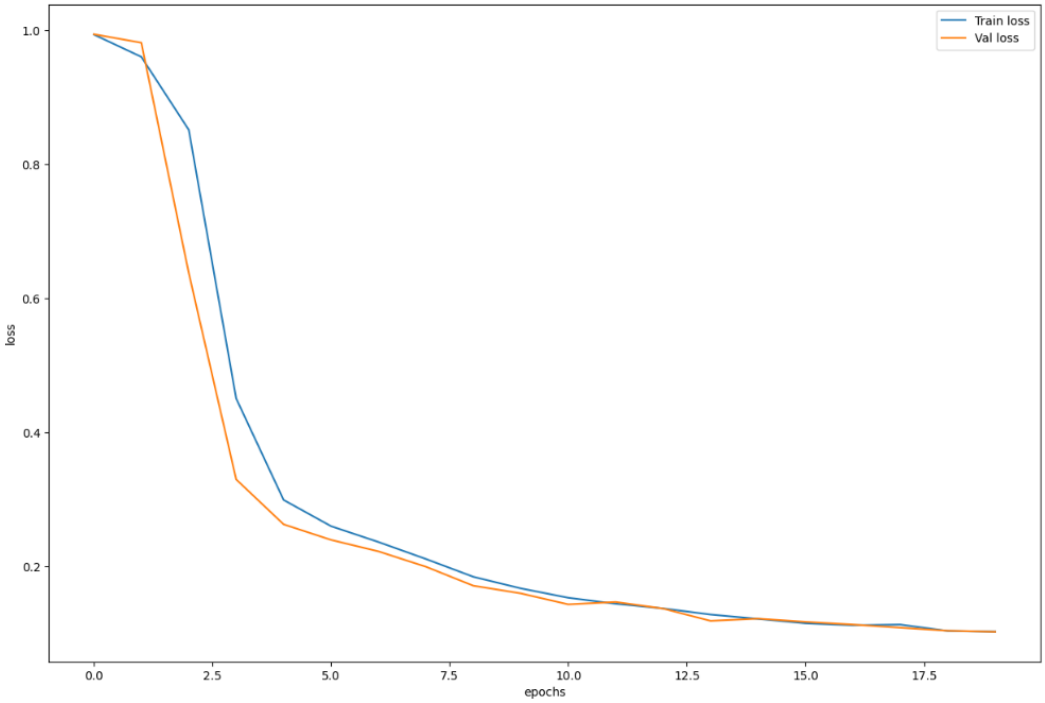
The SSIM loss function minimizes the SSIM index, thus the loss formula is the following:

$$\mathcal{L}_{\text{SSIM}} = 1 - \text{SSIM}(x, y)$$

The approach with this loss function has produced the following results:



And the loss function was the following:



Since these last results were overall better with respect to the ones produced using the MSE loss, we decided to use the SSIM loss in the final model.

3.1.2 CNN Variatonal Autoencoder

Another significant attempt regarding the image reconstruction was to modify our Autoencoder to obtain a Variational Autoencoder, so that we can take advantage of the probabilistic distribution of the latent space to make it more regular and interpretable. Together with the structure of the Autoencoder, we had to change also the loss function to adapt it to the new model:

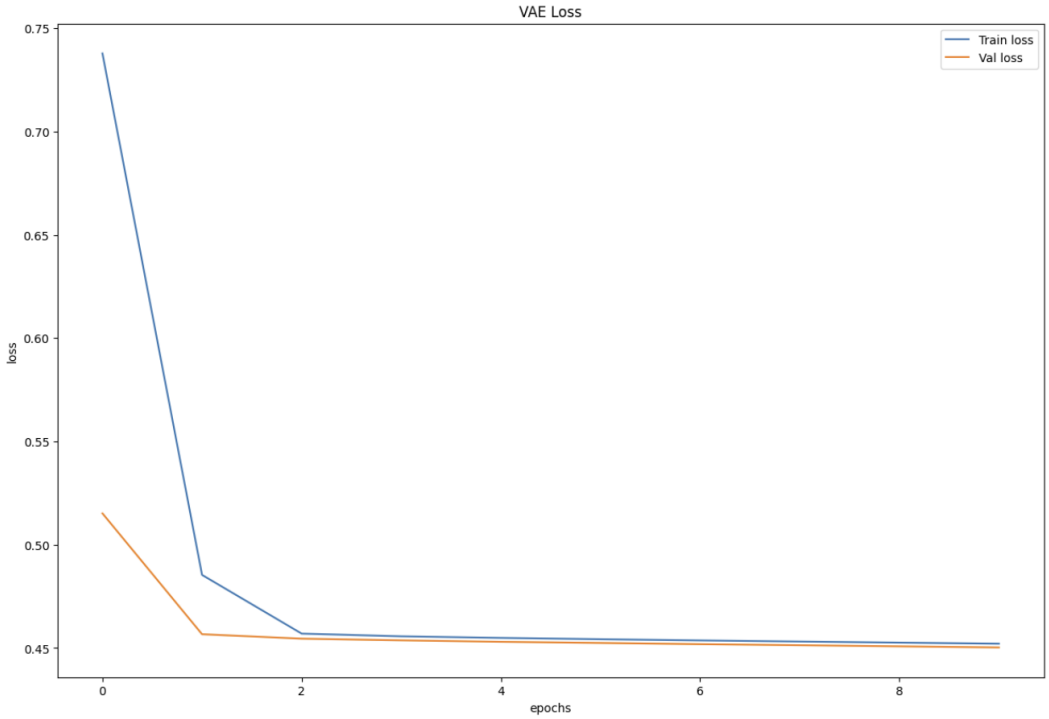
$$\mathcal{L}_{VAE} = \mathcal{L}_{ImageReconstruction} + D_{KL}$$

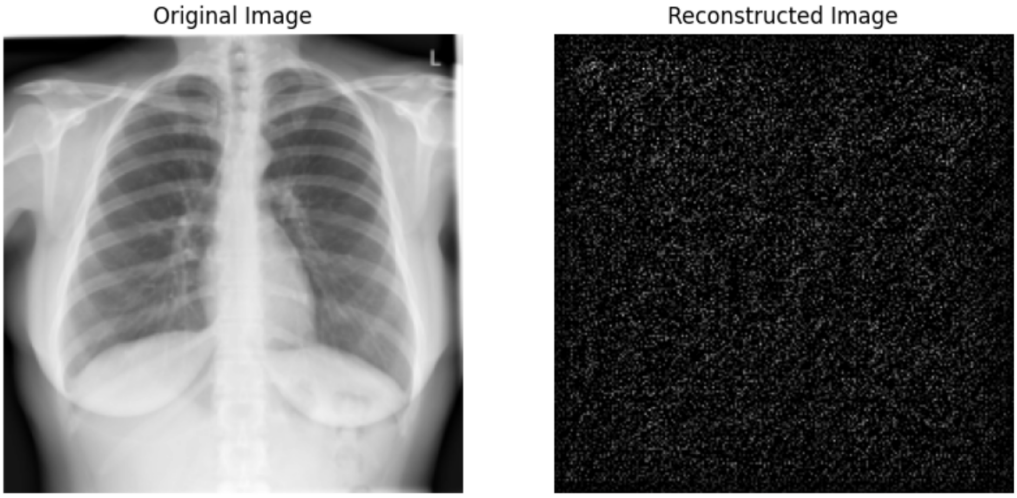
where $\mathcal{L}_{ImageReconstruction}$ is the SSIM loss already used in the basic Autoencoder version for quantify the similarity between the input X-ray and its reconstructed output, while D_{KL} is the *Kullback-Leibler* divergence, a value commonly used in the loss functions of VAE models:

$$D_{\mathrm{KL}} = -\frac{1}{2}\sum_{i=1}^{d}\left(1 + \log\sigma_i^2 - \mu_i^2 - \sigma_i^2\right)$$

During the first trainings, we observed that the KL divergence was too big. This phenomenon caused the loss to explode.
To resolve this problem, we attempted to use a β-VAE model, a variation of the VAE model which introduces a new static value β used to adjust and weigh the KL Loss. This value was found through the application of the *KL-annealing*, a technique that dynamically adjusts the value of β throughout the training process.
With the *KL-annealing*, we deduced a value of β that permits the loss function to co-exist with the KL loss, in this case 0.000001. In combination with this value, we employed:

- a dimension for the latent space in the VAE of 128, used for managing the images, their dimensions and their details in a more effective way.
- a learning rate of 0.0001, used especially to not have a too big KL divergence value during the training.





Despite this approach, the results of the VAE are far worse than those of the basic autoencoder, probably because of the structure of the autoencoder itself. In order to have a better result, the whole structure should be changed, and since that was not our main goal, it was decided to use the encoder of the basic Autoencoder for the final model.

3.2 Text Transformer

We used the text transformer for the text generation. Practically, the transformer takes as input the latent space generated by the image encoder and adapted by the mapper and the corresponding attention mask (tensor used in transformer models to control which tokens the model should pay attention to during the computation).
For this task, we used two pre-trained transformers:

- **GPT2**, a *decoder-only* transformer which takes as input embeddings of size 768;
- **BioGPT**, that is a variant of GPT2 trained on medical dataset. It's an *encoder-decoder* transformer (more complex than GPT2) and takes as input embeddings of size 1024.

We chose these two models mainly for their lightness and easy integrability.

We also tried to use the following models, but without success (difficulties in integrating them with our mapper):

- **OPT-125M**;
- **Pythia-70M**;
- **LLaMa3** and **TinyLLaMa**.

From the results, we noticed that:

- GPT2 is better at capturing the context of the dataset than BioGPT, but the generated text still remains disconnected from the dataset's reports;
- BioGPT is able to construct texts with meaningful sentences more easily than GPT2. The text concerns the medical field but is still far from the scope of the dataset.

In both cases, the generated text is of poor quality. Hence, we tried to fine-tune the transformer: we made a light training phase to make the transformer catch the main concepts of our dataset. At a first trial, we ended up in the so called **catastrophic forgetting**, the case in which the transformer does not learn anything from our dataset and it also *forgets its previous knowledge base*, regressing to an untrained phase. Eventually, we were able to fine-tune GPT2: to avoid catastrophic forgetting, we updated only the weights on the last two layers and the head (output layer). By doing so, we obtained a slightly worse transformer than the pre-trained one. So fine-tuning didn't help us a lot.

3.3 Latent Space Mapper

The **Latent Space Mapper** is a Feed-Forward Neural Network whose purpose is to map the latent space generated by the encoder to the embedding that the transformer needs in input in order to generate a valid medical report. It is without any doubt the core module of our architecture, but also its bottleneck: given the limited resources (time and computational power) we have, the

small size of the dataset and the fact that we rely on pre-trained transformers for the actual text generation, it was not possible to obtain optimal results to be used in a real case scenario.
However, we did our best to show that this idea (if scaled properly) could lead to good results.
For this module we proposed two different working approaches:

- **embedding approach**, which relies on computing the dissimilarity between the embedding of the real text (*ground truth*) and the embedding of the generated text (*prediction*)
- **token approach**, which makes the transformer generate the logits (*prediction*) of each word in the vocabulary, using them as a probability distribution, and then confront it with the tokenized real text (*ground_truth*)

For both approaches, the structure of the FFNN is the following:

```
class FF_mapper(nn.Module):

    def __init__(self, dim_input, dim_output):
        super().__init__()
        self.ff = nn.Sequential(
            linear_layer(dim_input, 640),
            linear_layer(640, 896),
            linear_layer(896, dim_output, last=True),
            nn.LayerNorm(dim_output)
        )


    def forward(self, latent_space):
        batch_size, C, H, W = latent_space.shape
        latent_space = latent_space.permute(0, 2, 3, 1)
        latent_space = latent_space.view(batch_size, H * W, C)
        return self.ff(latent_space)
```

Where "linear_layer" is a custom function that generates a pytorch Linear layer with the specified input and output dimentions, and applies (for all layers

except the last) a ReLU and a Dropout layer.
To train this model (in both approaches) the idea is using the transformer as if it was the last layer of the FFNN: in this way the module "FFNN + transformer" takes as input the latent space produced by the encoder and gives in output the generated text. Then, by computing the loss we let the gradients flow through the transformer and the FFNN, and in the end we update only the weights of the FFNN with the optimizer.

3.3.1 Embedding approach

As said before, in the embedding approach we compare the embedding of the real text with the embedding of the generated one.
In the first attempts to train this model we used MSE as loss function, but this resulted just in a convergence of the magnitude of the generated embedding to the real embedding (i.e. the scale of the two vectors were getting closer but their direction remained unchanged, thus the sentences were completely out of the medical context). To obtain the context we were looking for, we used a similarity index called "cosine similarity" (which indicates the alignment between two given vectors), whose formula is the following:

$$\text{CosSim}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$$

To use it in the loss function we needed to minimize this index, thus we used the following:

$$\mathcal{L}_{\cos} = 1 - \text{CosSim}(\mathbf{x}, \mathbf{y})$$

In later versions we created a custom loss function that combined the MSE with the Cosine Similarity, that was implemented through the following parameterized formula:

$$\mathcal{L}_{comb} = \alpha * MSE + (1 - \alpha) * \mathcal{L}_{cos}$$

The idea was to not only align the direction of the two vectors, but also their magnitude (in order to have results as precise as possible).
Unfortunately, despite all the combinations of α that we tried (and the few

epochs of training), the best result were given for $\alpha = 0$ (i.e. precisely the Cosine Similarity loss described before). Thus this was used in the final approach.

Training issues

It is important to highlight a problem that arised while training this module: in the first attempts the loss was not decreasing through the epochs.
The reason was that, during backpropagation, the gradients were not flowing (i.e. were equal to 0) from the transformer to the FFNN, resulting in a constant loss value for all the epochs.
After a long debugging session, we understood that the problem was related to the way we were generating text: we were using the **generate** function of the transformer (GPT-2 from Hugging Face) to directly get the generated text that would then be converted to an embedding and compared to the embedding of the real text. The problem with this procedure is that the *generate* internally uses **non-differentiable** functions as *argmax* and *sample*. This caused the gradients to be equal to 0, thus the FFNN was not learning.
To fix this issue we implemented the following function:

```
def soft_generate(inputs_embeds, attention_mask, labels,
    temperature=1.0):
    outputs = transformerModel(
        inputs_embeds=inputs_embeds,
        attention_mask=attention_mask,
        labels=labels,
        return_dict=True
    )
    logits = outputs.logits
    soft_tokens = nn.functional.softmax(logits / temperature,
    dim=-1)
    return soft_tokens
```

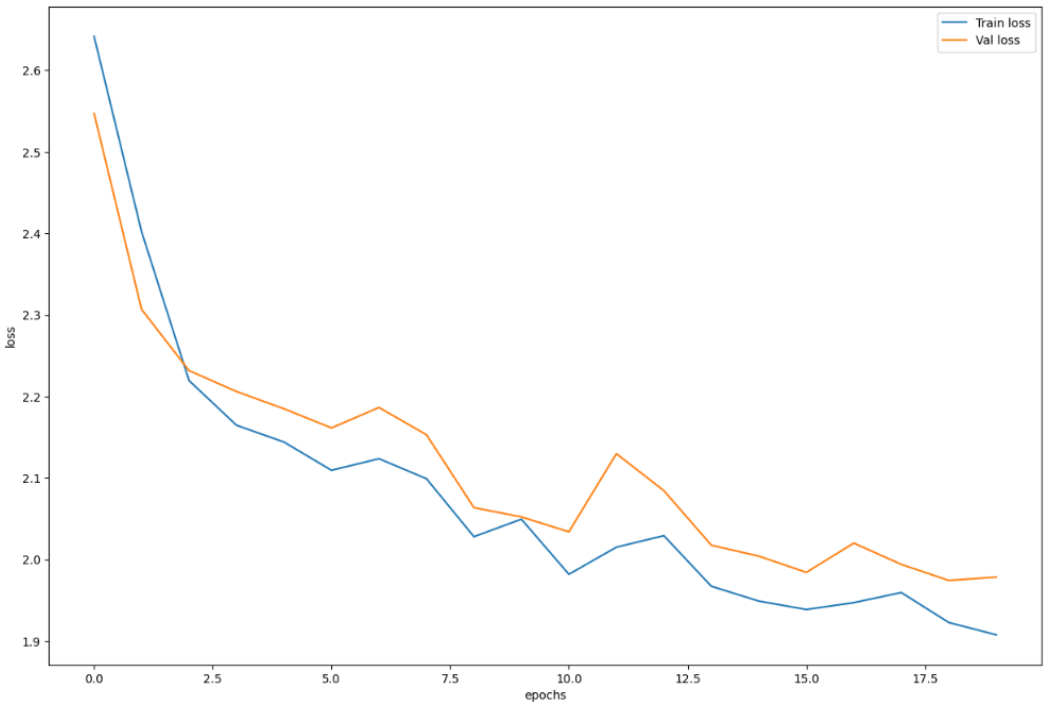
This function uses the forward pass of the transformer (that is differentiable) to extract the generated logits. Then, it applies a Softmax to get the related probability distribution over the vocabulary.

Eventually, we obtain the related embedding with the following:

```
pred_y = torch.matmul(probability_distribution,
    transformerModel.transformer.wte.weight)
```

that multiplies the probability distribution with the weight matrix of the embedding layer (i.e. transformer.wte.weight) to get the embedding of the most probable next word according to the probability distribution.

The loss function for this approach is the following:



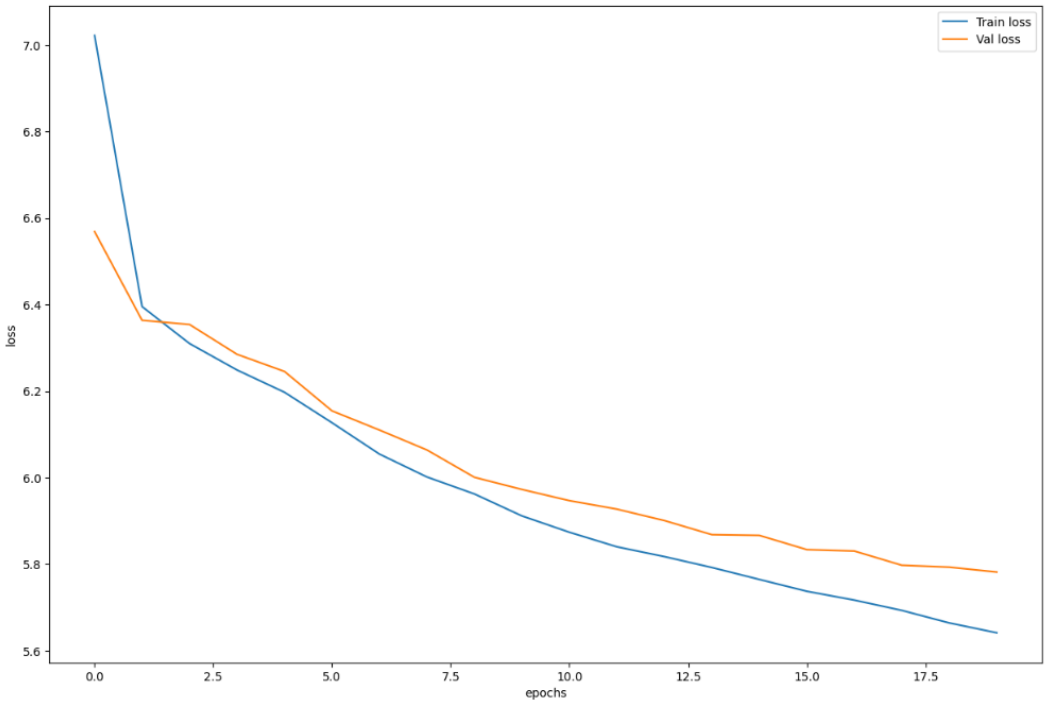
3.3.2 Token approach

As described before, in the token approach we train the model by making the transformer generate logits and then use them to obtain a probability distribution over the vocabulary. Then, we evaluate this probability distribution knowing which is the right word in the real text.

To do so, we use as loss function the Cross-Entropy Loss implemented by pytorch, which uses the following formula:

$$\mathcal{L}_{\text{CE}} = -\log\left(\frac{e^{z_y}}{\sum_{j=1}^{C} e^{z_j}}\right)$$

Which is the negative log-likelihood of the correct class after applying Softmax. It is important to highlight that in this implementation the Softmax function is already used internally, thus it is not necessary to apply that beforehand to the logits in order to get a proper probability distribution.
The loss function for this approach is the following:



3.3.3 Statistical analysis

To better understand whether the two approaches worked properly or not, we also computed a statistical analysis by comparing the expected embeddings of the transformer to the ones generated by the mapper.
We made all the calculations on a dataset sampling and computed the predicted and real embeddings in the function *collect embeddings*.
Once we have collected the predicted and real embeddings, we start by normalizing them, using the **L2 normalization** (*euclidean norm*) and then we calculate some statistics:

- **mean and standard deviation** for both real and predicted embeddings. These will be arrays, since they represent measurements for each

dimension of the embedding (in the case of GPT2, the embeddings have dimension 768);

- **average cosine similarity** between the embeddings;
- **average euclidean distance** between the embeddings;
- **mean magnitude** (computed by L2 normalizing the mean array previously calculated) and **mean standard deviation** (computed by averaging the standard deviation array already calculated).

Embedding Approach using GPT2:

- **Average Cosine Similarity**: 0.3015
- **Average Euclidean Distance**: 1.1727
- **True Embeddings - Mean Magnitude**: 0.7554 (Std: 0.0220)
- **Predicted Embeddings - Mean Magnitude**: 0.5927 (Std: 0.0269)

Token Approach using GPT2:

- **Average Cosine Similarity**: 0.3318
- **Average Euclidean Distance**: 1.1438
- **True Embeddings - Mean Magnitude**: 0.7554 (Std: 0.0220)
- **Predicted Embeddings - Mean Magnitude**: 0.6844 (Std: 0.0240)

Embedding Approach using BioGPT:

- **Average Cosine Similarity**: 0.1035
- **Average Euclidean Distance**: 1.3191
- **True Embeddings - Mean Magnitude**: 0.7304 (Std: 0.0189)
- **Predicted Embeddings - Mean Magnitude**: 0.2781 (Std: 0.0294)

1. **Cosine Similarity** measures how similar two vectors (the embeddings) are in direction, ignoring their magnitude (length). It is in the range $[-1, 1]$, where 1 means the vectors are identical in direction, 0 means they are orthogonal (no similarity), and -1 means they are diametrically opposed. So in our case:

    - using GPT-2 (for both the approaches), values ~ 0.3 suggest weak but non-random alignment. Hence, the mapper seems to be able to capture some of the structure of the embedding space, but not enough to produce high-quality text.
    - Using BioGPT, the average cosine similarity drops significantly to 0.1035, indicating poor alignment in direction. This suggests that adapting the latent space for BioGPT is more challenging, probably due to the more specialized and structured embedding space of BioGPT compared to GPT-2.

2. **Euclidean Distance** measures the absolute distance between two points (vectors) in space. Lower distance indicates that the embeddings are closer together, while higher distance indicates they are further apart. It is in the range $[0, \infty)$, where 0 means the vectors are identical and larger values indicate greater dissimilarity. In our case:

    - For GPT-2 (for both the approaches), the Euclidean distance is approximately 1.14–1.17, while for BioGPT it increases to 1.3191.
    - The higher distance for BioGPT confirms that the mapper struggles more to replicate the embedding space accurately when dealing with BioGPT.

3. **Magnitude** represents the length (or norm) of an embedding vector. The greater the difference between the real embeddings magnitude and the predicted ones magnitude, the bigger the discrepancies between the two embeddings (real and predicted). In our case:

- With GPT-2, predicted embeddings have a mean magnitude deficit of about 20% in the embedding approach (true embeddings have mean magnitude of 0.7554 and predicted embeddings have mean magnitude of 0.5927) and 9.4% in the token approach (true embeddings have mean magnitude of 0.7554 and predicted embeddings have mean magnitude of 0.6844) compared to the true embeddings. So the token approach seems to be slightly better than the embedding approach. This indicates that the mapper is able to produce embeddings that are somewhat aligned with the true embeddings, but still not powerful enough to generate high-quality text.
- With BioGPT, the predicted embeddings have an even larger magnitude deficit, being approximately 62% lower than the true embeddings (0.7304 for true embeddings against 0.2781 for predicted embeddings). Also in this case BioGPT behaves worse than GPT2.

We also built some plots about the cosine similarity distribution and the euclidean distance distribution between the two embeddings (real ones and predicted ones).

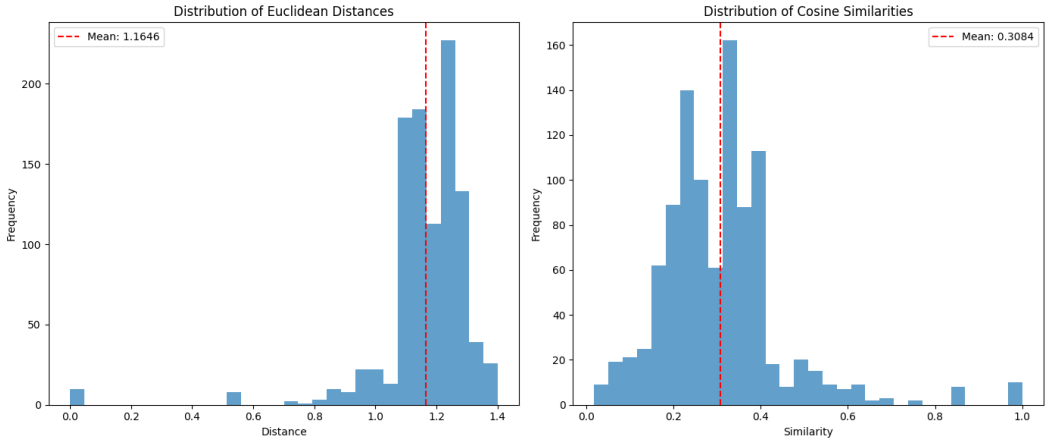


Figure 3.2: Embedding approach using GPT2.

As you can see from the plots (Figures 3.2, 3.3, 3.4), the distributions of the cosine similarities and euclidean distances have *bell-shaped* (Gaussian-like) distributions, meaning that most samples have similar behavior around a central

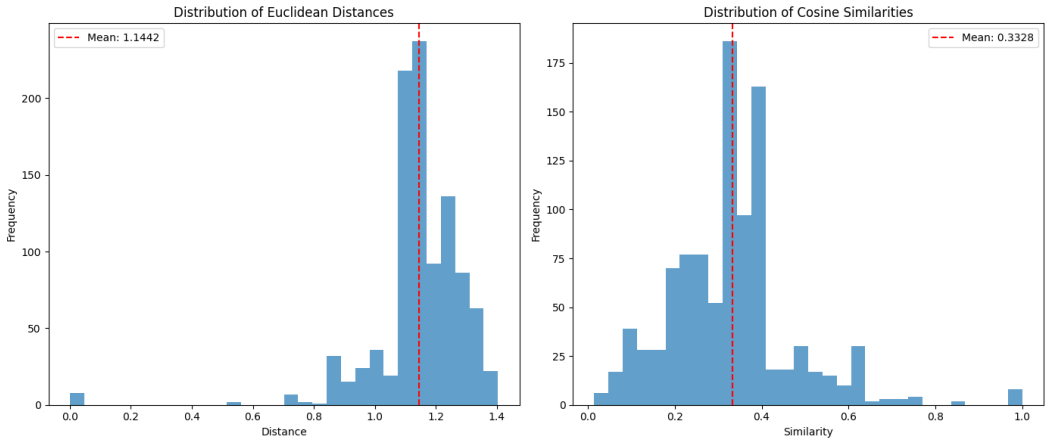


Figure 3.3: Token approach using GPT2.

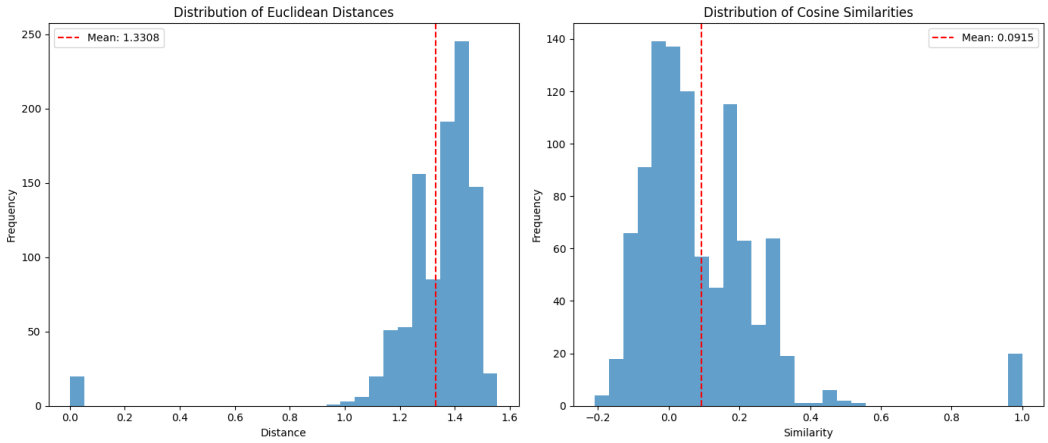


Figure 3.4: Embedding approach using BioGPT.

average value. In other words, the mapper's performance is consistent across the dataset.

Eventually, we applied 3 dimensionality reduction techniques to better understand and visualize the difference between the embeddings:

- **PCA** (*Principal Components Analysis*): it is a statistical technique used for dimensionality reduction that transforms high-dimensional data into a lower-dimensional representation while preserving as much of the original variance as possible. It is composed of the following steps:

  1. standardize the embeddings;
  2. calculate the covariance matrix and its eigenvectors and eigenvalues (which respectively represent the directions of the principal components and the magnitude of the variance along those directions);
  3. data projection onto the selected principal components, resulting in a lower-dimensional representation.

  PCA tries to preserve as much variance as possible in the first few components (it preserves large-scale relationships, the global structure). However, there can be loss of information due to the dimensionality reduction. We can check PCA accuracy by paying attention to the **explained variance** variable: it is an array where each element represents the proportion of the total variance in the original data that is *"explained"* by the corresponding principal component. It basically represents how much information or variance each direction (or principal component, that are PC1 and PC2) captures from the original data.

  Moreover, if the data is not linearly separable, PCA may miss important structure. For this reason, we implemented also 2 non-linear methods (Umap and t-SNE).

- **Umap** (*Uniform Manifold Approximation and Projection*): it is a powerful *non-linear* dimensionality reduction technique. It is composed of

the following steps:

1. *constructing a high-dimensional graph.* Umap first tries to understand the relationships between the data points in the original high-dimensional space. For each data point, it identifies its nearest neighbors (the number of neighbors is controlled by the *n-neighbors parameter*). It then creates a weighted graph where edges connect each point to its neighbors. The weight of an edge reflects how "close" the two points are in the high-dimensional space.
2. *Approximating the manifold.* The graph constructed in the previous step is used to approximate the local structure of a manifold (complex, curved surface) around each data point.
3. *Defining a Low-Dimensional Graph.* Umap then creates a similar graph in the low-dimensional space (2D). It starts with a random initial layout of the points in the low-dimensional space.
4. *Optimizing the Low-Dimensional Layout.* Umap uses an optimization process (similar to gradient descent) to adjust the positions of the points in the low-dimensional space with the goal of minimizing the difference between the structure of the high-dimensional graph and the low-dimensional graph. It tries to pull points that were close in the high-dimensional space closer in the low-dimensional space, and push points that were far apart further away. The *min-dist parameter* plays a role here by setting a minimum allowed distance between points in the low-dimensional embedding.

This technique tries to balance local and global structures.

- **t-SNE** (*t-Distributed Stochastic Neighbor Embedding*): another non-linear dimensionality reduction technique. It is composed of the following steps:

  1. *probability distributions.* t-SNE converts high-dimensional distances between data points into conditional probabilities that represents

the likelihood of points being neighbors. It then attempts to recreate a similar probability distribution in the low-dimensional space.

2. *Minimizing divergence.* t-SNE tries to make the distances between points in the low-dimensional space reflect their relationships in the high-dimensional space.

3. *Non-linear mapping.* It allows to capture complex relationships and structures in the data.

An important variable for t-SNE is the **perplexity**: it corresponds to the number of nearest neighbors considered for each point. Lower perplexity values focus on local relationships, while higher values consider global relationships.

Anyway, t-SNE focuses very strongly on preserving local neighborhoods, sacrificing the global structure.

In short, with these methods we are reducing the size of the embeddings from 768 (default size for GPT2) to 2. By doing so, we can plot the embeddings on a cartesian plane, but we could lose information due to the dimensionality reduction! For this reason, we use them only for visualization. For more accurate comparison, we used the statistics discussed above (mean, standard deviation, cosine similarity, euclidean distance).
Below we show the plots for the three dimensionality reduction techniques:

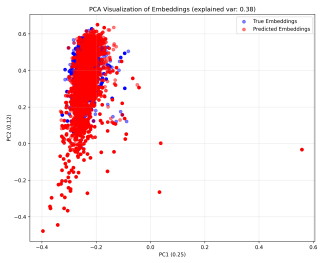


Figure 3.5: Embedding appr. on GPT2

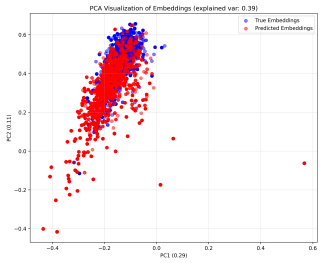


Figure 3.6: Token appr. on GPT2

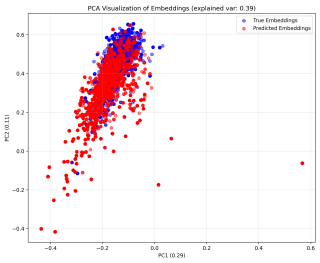


Figure 3.7: Embedding appr. on BioGPT

Figure 3.8: PCA plots

For PCA, in all the plots (Figures 3.5, 3.6, 3.7) the predicted embeddings are overlapped to the real ones, with a better performance for the embedding approach using GPT2 (quite perfect embeddings overlapping). Anyway, we should also consider the low explained variance of the PCA (around 0.38-0.39), which means that the PCA is not able to capture the whole variance of the data, so we should be careful in interpreting these results and rely also on the other used techniques.

In the Umap calculation, we used for the parameter n-neighbors the values 5, 15, 30. They represent the number of nearest neighbors considered for each point. Lower values focus on local relationships, while higher values consider global relationships.
The min-dist parameter was set to 0.1. Here we'll display the results for n-neighbors = 5, 15 and 30:

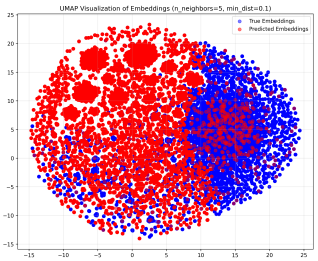


Figure 3.9: Embedding appr. on GPT2

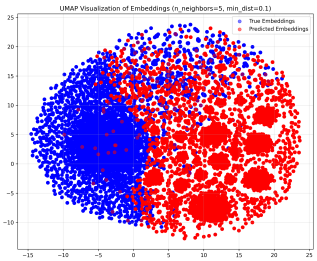


Figure 3.10: Token appr. on GPT2

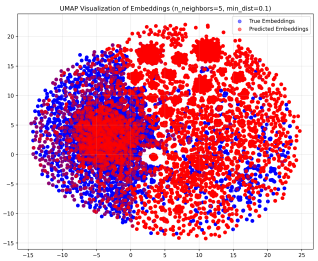


Figure 3.11: Embedding appr. on BioGPT

Figure 3.12: Umap using n-neighbors = 5.

As you can see (Figures 3.9, 3.10, 3.11, 3.13, 3.14, 3.15, 3.17, 3.18, 3.19), in this case BioGPT behaves better than GPT2 with n-neighbors = 5, since there's a better overlapping between the embeddings; while for n-neighbors = 15 and 30 the best perfomances were obtained by the embedding approach on GPT2 and BioGPT. However, all the models seems to work better on n-neighbors = 15 and 30: this means that the models can capture and learn global patterns more easily than local patterns (n-neighbors low).

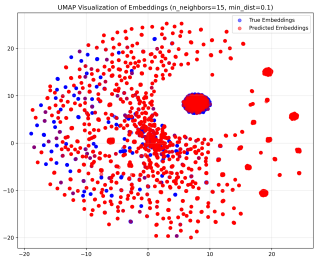


Figure 3.13: Embedding appr. on GPT2

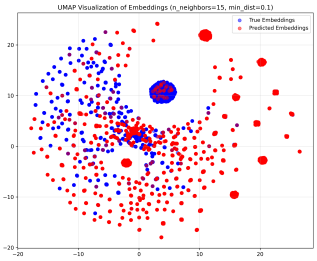


Figure 3.14: Token appr. on GPT2

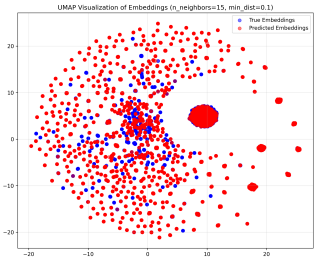


Figure 3.15: Embedding appr. on BioGPT

Figure 3.16: Umap using n-neighbors = 15.

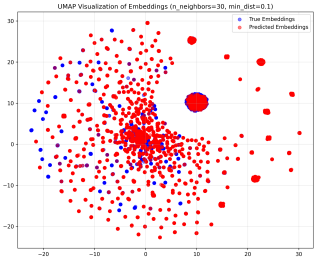


Figure 3.17: Embedding appr. on GPT2

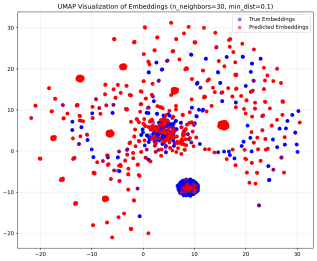


Figure 3.18: Token appr. on GPT2

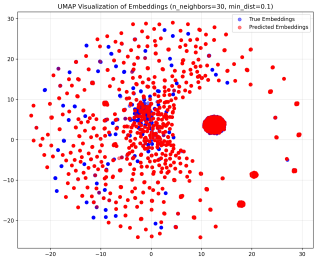


Figure 3.19: Embedding appr. on BioGPT

Figure 3.20: Umap using n-neighbors = 30.

In the end, we report also the t-SNE results. Since it can be computationally expensive for high-dimensional data, we decided to apply first PCA (to reduce dimensionality to 50 components) and then t-SNE on the PCA results. We computed t-SNE with perplexity equals to 5, 30 and 50 (Figures 3.21, 3.22, 3.23, 3.25, 3.26, 3.27, 3.29, 3.30, 3.31):

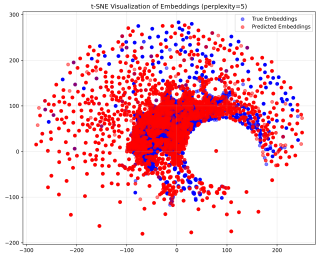


Figure 3.21: Embedding appr. on GPT2

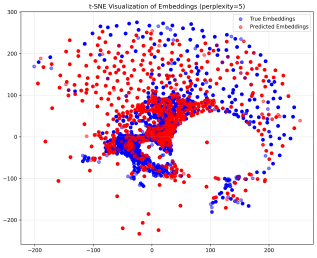


Figure 3.22: Token appr. on GPT2

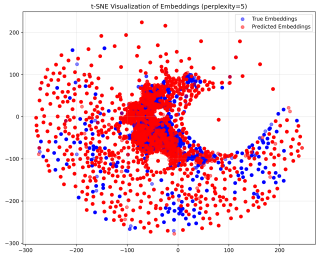


Figure 3.23: Embedding appr. on BioGPT

Figure 3.24: t-SNE using perplexity = 5.

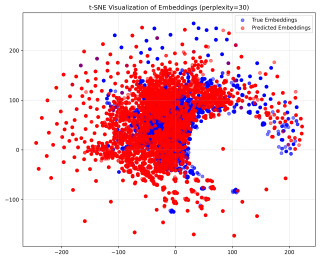


Figure 3.25: Embedding appr. on GPT2

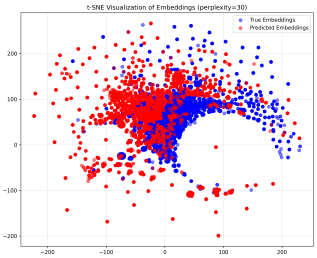


Figure 3.26: Token appr. on GPT2

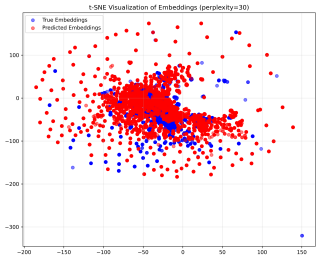


Figure 3.27: Embedding appr. on BioGPT

Figure 3.28: t-SNE using perplexity = 30.

As for the Umap, also in this case the embedding approach (on both GPT2 and BioGPT) looks the best, with a pretty good overlapping (on both perplexity equals to 5 and 50). And between GPT2 and BioGPT, the first one seems quite better.

Hence as an overall result of this statistical analysis, we can safely say that the best model configuration found is the embedding approach used on GPT2.

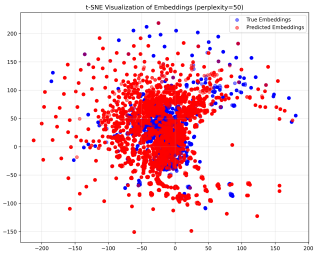


Figure 3.29: Embedding appr. on GPT2

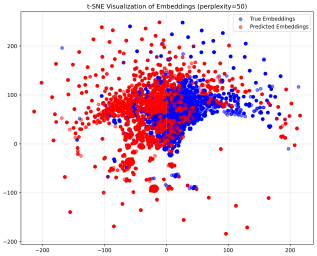


Figure 3.30: Token appr. on GPT2

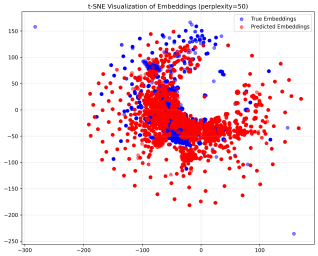


Figure 3.31: Embedding appr. on BioGPT

Figure 3.32: t-SNE using perplexity = 50.

Chapter 4

Results

In the end, after comparing the generated texts we chose the GPT2 model (without fine-tuning) to be the best one.
Here we show some examples of the generated texts:

```
Decoded generated text: ., and the, the the.
.. and,,. The first, it's a new, "The first. It's not a good news. I'm not the first time, I have a lot of peopl
e have been told me to the press release, but I don't know for the fact that the number of patients with a histo
ry of the disease, or the type of cancer, we have no known risk factors for cancer. This is a major cause of dea
th in-person interviews with the United States. (A) The following are the following:

1.1) A recent study has shown that a large number, including the incidence of heart disease and/or heart failure
of aortic valve disease. 2.2
 (C) In the past, there have we've seen a number and a small number (1 in the last two years, three of these pat
ients have died from the US Food and Drug Administration (FDA) and in some cases of acute myocardial infarction,
heart attack, stroke, liver failure, kidney failure or other conditions. 3.5
, 4.3% of all patients who have had a heart condition that ischemic stroke. Injuries are rare, non-obese patient
s. There are no evidence that there is any evidence of any of this disease to be considered asymptogenic agents
are not yet known to have any known side effects. 4)
The following is the present study, a study of human papillomatosis is associated with diabetes mellitus. A stud
y in a patient with type 2 diabetes.0.6
"
```

Figure 4.1: Embedding approach with GPT2

Notice that the texts generated by the token approach (Figures 4.2 and 4.4) are the semantically worst ones, while the text generated by BioGPT (4.5) is the best from a semantical point of view and looks like a medical report but is not related to the dataset.

We can see also that the texts generated by the fine-tuned GPT2 are not so different from the ones generated by the corresponding pre-trained one. Moreover, the embedding approach model with fine-tuned GPT2 (4.3) is able to capture words which look much related to the dataset context (*heart, lung*).

```
Decoded generated text: . the the, the.. of the of.,. is. or the is,, and. (.
. and, is the

, of, a. The other side of a side effect of
The side effects of The side is a sides of it's a part of itself. a whole side. A side-effect of its own. side a
half a piece of land. Side effects a small piece. It's side affects the whole of another. There's another side a
ffect the other. Another side affected by a line of water. This side has a large part to the left. In this side,
another is it is not a good time to left and right, it appears to right and left, left is left to and is is ther
e is right to, right is and and to is an is to are and the right side and without, are left side to a left- and
are the eye and eye is front and front, front is appears and appears, there are there and two, both are is has a
nd both, as is without and a right- is both is two and an eye, two is one and there, without a second, an and se
parate, or is in the face and face, in, we are, with and see and look, eye. left eye-


 is face is present and in and canary, see,
 and visible, face. are. eye are two.

. front. face of is of are are in. in are a, this is are of eye level. there. it. see. this, one, look. look of
left in is visible. ". to. by the look is-. with. can. do. right. have. appear. present. one. an. we see of
```

Figure 4.2: Token approach with GPT2

```
Decoded generated text: . the is.. The. No. no. is the. the the, the no no size. and no right. There is no evide
nce. in the normal. normal, normal is normal and normal are normal in size, no, size and size is, is within the
limits. Heart. within. eff. size of the size or size are no space. limits of size limits, limits are the right,
heart.
 The size in normal size no clear no heart, lung, and heart and lung. heart is heart the heart of heart are hear
t eff is clear. Card size within limits is in no the space is size the is space within normal within within, wit
hin and within size The heart in is card heart no is present. space no in. are. of no limits the within is of sp
ace are limits within no within space, space and space space limits no and limits in space of normal limits and
the air. clear space in, are size No space the in are within or within of is is limits limits space or no of lim
its ple size space normal the and in limits or limits right of. air, air within in air is air are space eff spac
e The space size normal or in within are of within air space with no air in and. or. right space air no or space
right is eff, in eff size eff the eff no eff in right the left., right and right are clear within eff within rig
ht no are, or right in left is and is right air eff are in in or eff eff and eff right within left eff air the a
re eff of right eff or the pulmonary space ple is or is left space for space No eff limits The no normal no left
, left and left no ple. ple, eff normal space at heart within a size size card size
```

Figure 4.3: Embedding approach with fine-tuned GPT2 (10 epochs)

```
Decoded generated text: ... the the. of The No There, of. No.,. orX is
 or. The. There is. is the the No No The No, the The The There.
. noThe No There No no. eff. inNo no No
 No eff eff No the no no eff and in. This is no the is No is The no The is is There no is eff The the, No are no
, no There are are. and No in the eff is, There There the right. are No right, The eff, eff no

 There The, is in is
 The
,
 is of is and. within the There, are the and is within eff the
X. sizeicAxor. space This within.X is or is are is normal. ple ple. pneum. normal is ple is space. cont. withoth
i1 a cont pneumThere

 size. A with. In normal, and the ple, or the size is there right ed. eda for atural. right is size, ple No and
no and The and eff and, in, size and


 eff There
 the are, normal space is right no are The size No or no or No of the normal and are There and normal the in no
of,, within is left. left, right and in The are are eff or, of no in eff are and or The or eff in and size of ef
f of The in No space, space and within, clear. a. at. there. limits. by. focal. Right
```

Figure 4.4: Token approach with fine-tuned GPT2 (10 epochs)

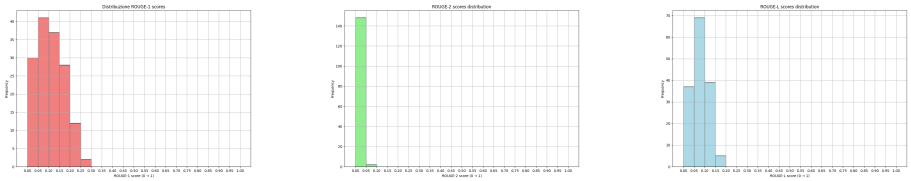
```
Decoded generated text: The Effect of the Addition of a New Class of Antioxidants on the Growth of A Novel Appro
ach to the Treatment of Chronic Obstructive Pulmonary Disease. Chronic obstructive pulmonary disease (COPD) is a
progressive disease that is characterized by airflow limitation and inflammation. (1) The disease is associated
with a high morbidity and mortality. [...]. "[This corrects the article DOI: 10.2196 / resprot.7008.] In Vitro a
nd In Vivo Evaluation of Anti-Inflammatory and Antioxidant Activities of Essential Oil from Essential oils (EOs)
are a group of natural products with diverse biological activities. 'Erratum: Corrigendum to" The Role of Vitami
n D in the Prevention of Cardiovascular Disease: A Systematic Review and Meta-Analysis. A Narrative Review. *. J
. Clin. Med. 2017, 18 (3), 531.. DOI. 10.1177 / 018735917717611861751 9. The authors would like to note that the
name of "Dr. José Maria" was incorrectly spelled as "Józef Márquez.," and the correct name is "Mario Mórczález"
[corrected] The above article has been corrected.). • The name "David Marian" should be "Astonishi Mazzaro.;" "B
ángáló" is the first name for the author. ¿"Dr Jos Mariarzo" has not been included in this study.}. +. Vitamin D
3 is an important component of calcium and phosphate metabolism. < / p >.\. $\ rm {\ mathrm.br. > < m
```

Figure 4.5: Embedding approach with BioGPT

However, we consider the embedding approach model with pre-trained GPT2 the best overall (based particularly on the generated text).
To have a better understanding of the final performance of the main model, we introduced two famous metrics used specifically to evaluate the generated text:

- **ROUGE** (Recall-Oriented Understudy for Gisting Evaluation) measures how much the generated text overlaps with the reference text.
  - ROUGE-N: Compare n-grams (sequence of N words) between generated and reference texts.
  - ROUGE-L: Measures the Longest Common Subsequence (LCS) between generated and reference.
- **BLEU** (Bilingual Evaluation Understudy), which compares how many n-grams (words or sequences of words) from the generated text match those from the reference (correct) text.

These two metrics were used on a random sample of the test set to evaluate the overall performance of the model.



(a) ROUGE-1 results (b) ROUGE-2 results (c) ROUGE-L results

Figure 4.6: ROUGE metrics results

The results of the ROUGE metrics aren't that good, probably because the metric does not take into account the overall meaning, logical coherence, or grammatical correctness.

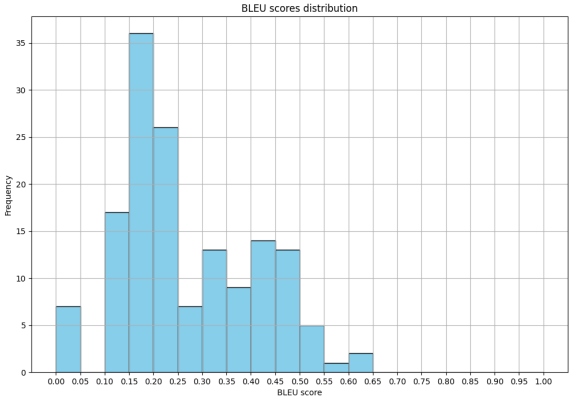


Figure 4.7: Model structure

In the other hand, BLEU provides better results than ROUGE. While both are n-gram-based metrics, BLEU evaluates the precision of the generated text in matching the reference output and applies a brevity penalty to short predictions. This makes it more suitable for tasks where the generated output is expected to follow a specific structure or use domain-relevant terminology, like the medical field in our case.

Chapter 5

Conclusions

In this project, we explored multiple models to address the given task of generating a text from the image encoding, carefully evaluating their performance using standard metrics. Among the models tested, the embedding approach model with GPT2 demonstrated the best overall performance (in metrics, statistics and text generation).

It is important to note that further improvements could be achieved with access to greater computational resources. In particular, training larger models and fine-tuning properly the pre-trained transformer.

Moreover, future work may also explore alternative architectures, like replacing the pre-trained transformer with a custom one or with a vision transformer.

Overall, this project provided valuable insights into the challenges and opportunities in the field of image-to-text generation, particularly in the medical domain. The results obtained highlight the potential of combining different models (CNN, VAE, Transformer, FFNN) and approaches to achieve better performance in generating meaningful and contextually relevant text from images.