

# CLIP Hackathon Task

## Advanced Topics in Deep Learning: The Rise of Transformers

Raul Singh  
Politecnico di Milano

raul.singh@mail.polimi.it

Davide Rigamonti  
Politecnico di Milano

davide2.rigamonti@mail.polimi.it

### Abstract

*This paper documents the iterative development and enhancements of a custom CLIP-based model for a biomedical application utilizing the ROCO dataset, culminating from a hackathon held for the Advanced Topics in Deep Learning course. Initiated with manually implemented encoder blocks and basic training/evaluation strategies, the project evolved into leveraging pre-trained models like ConvNeXt and BERT, aiming to optimize performance and mitigate overfitting issues. Significant enhancements with respect to the baseline model emerged from improvements in the model, the training pipeline, the evaluation metrics and the visualization techniques. Some of these improvements, such as lazy loading for the data pipeline, already got explored during the hackathon and were improved upon in the following time; other techniques, such as the use of pre-trained models and data augmentation, were added as novel improvements upon the base model. The main two tasks that have been tackled are image and text retrieval, however an additional zero-shot task in the form of image classification has been considered and showcased in order to demonstrate the model's adaptability. Evaluating the model's performance relied on custom metrics like Concept Overlap and Caption Equality, which were specifically crafted to enable easy understanding of the model's capabilities and simple comparisons with the task baselines. Conclusively, this project showcased a progressive transition from a barely functioning model, to an optimized architecture that performs fairly well according to the proposed evaluation metrics.*

### 1. Introduction

This is a project that has been developed in the context of the *Advanced Topics in Deep Learning: The Rise of Transformers* course held at Politecnico di Milano and supervised by Professor Matteo Matteucci. The initial assignment was carried out as a one-day hackathon held at Politecnico on the 1st of February 2023. Then, groups composed of M.Sc.

students were asked to improve upon their initial design in order to overtake their baseline performance by means of creative additions and overall improvements.

Groups were expected to implement their designs from scratch, without copy-pasting pieces of code from the assigned references. The language recommended for the implementation is **Python**, with the addition of the **TensorFlow/Keras** libraries.

We used **Jupyter Notebooks** as a way to organize our work into self-contained and meaningful subtasks, we utilized **Kaggle** and **Google Colab** as means to access computing power to train our models when local resources weren't available to us and lastly, we chose **L<sup>A</sup>T<sub>E</sub>X** to write the final report.

#### 1.1. Task

The central challenge for this project is to develop a functional implementation of the **OpenAI CLIP** model for the main task of **image retrieval** in a biomedical application. The following blog post [3] and research paper [2] were given in the assignment as a starting point for understanding the principles of the model, in addition to the information already given by Professor Giacomo Boracchi during the course's frontal lectures.

#### CLIP Principles

In short, **CLIP** stands for **C**ontrastive **L**anguage-**I**mage **P**re-training, and it is composed by two *encoders* (one dedicated to images and one for text) which are trained to minimize the loss computed as the *cosine similarity* of the embedded image pairs.

This peculiar pre-training technique enables the model to achieve relevant performance metrics on a wide array of zero-shot tasks (such as classification) while using a fraction of the dataset of other popular deep learning models performing the same task.

#### 1.2. Dataset Analysis

The dataset provided is a *simplified* version of the **ROCO** dataset (**R**adiology **O**bjects in **C**ontext) [1] which

consists of 83275 elements composed of:

- A low resolution image that has been reduced in size.
- A corresponding textual caption that describes the image.
- A set of concept labels that characterizes the attributes of the image.

There are 80813 unique captions as most of them are present in the database only one time, the most common caption appears in 261 images.

Images can have from 1 to 50 concepts associated to them from a total of 8374 concepts, averaging around 5 concepts per image.

Concepts can have from 2 to 25989 images associated to them from a total of 83275 concepts, averaging around 47 images per concept.

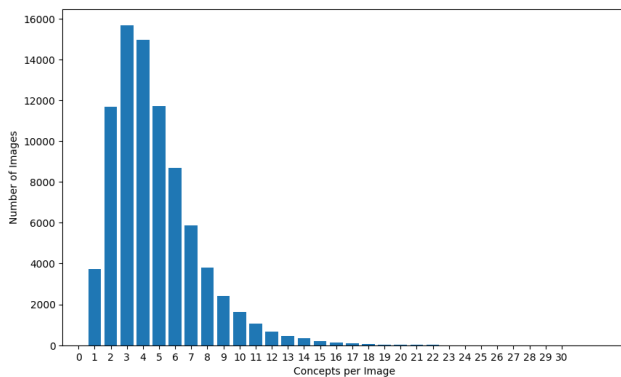


Figure 1: Concepts per Image

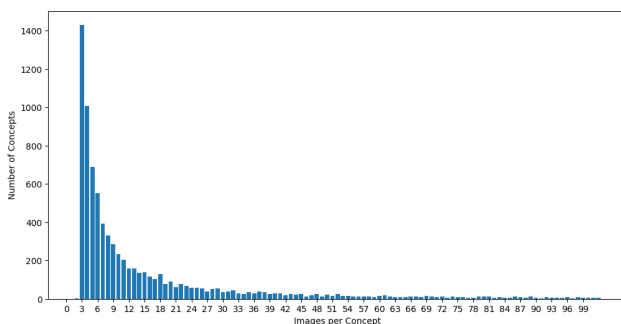


Figure 2: Images per Concept

## 2. Hackathon Results

The self-imposed goal for the hackathon was to build a CLIP model capable of performing image retrieval, thus we initially left out the portion of data regarding concepts and labels.

Unfortunately we didn't manage to create a functioning notebook in the allotted time. However, we can still consider ourselves satisfied with our work, since right after the event we noticed that we were just a small change away to obtain a fully working CLIP model.

### 2.1. Initial Intuition

During the hackathon, we opted for splitting our efforts into two separate tasks: model creation and data gathering. We created 2 preliminary notebooks that tackled the defined tasks, these notebooks were then joined together into a single notebook, considered the final product for the hackathon.

#### 2.1.1 Data Gathering

Since the given dataset was problematic to handle due to its dimension and its uncommon compression encoding, part of our efforts were dedicated to create a proper *data pipeline* to feed the model.

The preliminary data gathering pipeline loads captions and images separately, performing different preprocessing for both of them (using the *transformer custom standardization function* seen during lectures for textual data, while performing *resizing* and *normalization* for image pixel data). The captions are then vectorized using a **Keras text vectorization layer**, all the elements are split manually into train, test and validation sets and stored into Python lists.

Various visualizations of the imported images and data were added to ensure the correctness of the loading process.

#### 2.1.2 Model Creation

For the preliminary model we decided to implement the encoders from scratch in order to avoid adding useless complexity to our prototype.

For the image encoding we defined a simple CNN composed by 3 **convolutional layers** with increasing neuron count (from 64 to 256 neurons) with the inclusion of **batch normalization layers**. At the top of the convolutional section we added a **flattening layer** followed by two **dense layers** to obtain a final embedding of size 128. All of the hidden layers have **ReLU** activation functions.

For the text encoding we used a *transformer encoder block* that was presented during the course lectures, which makes use of **multi-head attention**, **layer normalization** and **dropout** layers; at the end of the block there are 2 **dense layers** to obtain a final embedding of size 128. Before being fed to the encoder block, inputs pass through a double **embedding layer** that implements *positional embedding* by addition.

The complete model routes the two inputs (image and text) to their respective encoders, normalizes the resulting embeddings and obtains the *logits* by applying the formula

outlined in the OpenAI paper:  $\text{logits} = (I_e \cdot T_e^\top)e^t$ . Then, the individual losses are calculated by means of **categorical cross-entropy** and are averaged to obtain the final loss.

## 2.2. Idea Refining

As mentioned before, the very last notebook obtained at the end of the hackathon didn't work correctly as there were some small discrepancies between the sizes of the expected inputs vs. the actual inputs, and we were trying to apply *categorical cross-entropy* utilizing numeric variables that were not *one-hot encoded*.

The day after the hackathon took place we dedicated some of our time to slightly refine the final product obtained in order to have a functioning model. By adapting the structure of our network to the input data, fixing minor conflicts related to typing and employing **sparse categorical cross-entropy** instead of *categorical cross-entropy*, we managed to obtain a working CLIP implementation.

Due to the inefficient loading technique we weren't able to hold the entirety of the dataset and the model in memory, thus we couldn't test it properly. However, by considering a fraction of the dataset and training the model on it for around 10 epochs, we noticed that the results were quite scarce as the loss would not get any lower after the first few epochs. When considering a top-k image-retrieval task on the training dataset with an exact caption as a query, the target image was only occasionally present in the top-10 matched results, and never in the top-5.

The reported results of this step are based on empirical observation as we did not have a proper evaluation system for this task yet. This is due to the fact that our main goal for the hackathon was to produce a functioning model, not evaluate its performance.

## 3. Post-Hackathon Improvements

This section is dedicated to exploring all the improvements (successful and non) that were performed in the months following the initial hackathon challenge. Some of these improvements are strictly related to the model structure and performance, while others are centered around *data exploration*, *data gathering* and *model evaluation*.

### 3.1. Loss Function

We decided to reimplement the loss function and how it is integrated inside the CLIP model from scratch as we previously made use of **TensorFlow lambda layers** to perform mathematical operations. We noticed that these layers were causing problems in both the gradient calculation process and during import/export operations.

The approach we took for the new loss was quite radical as we decided to delve into the **Keras model** by *overriding* the `train_step` and `test_step` in order to have more

control on how the model calculates loss and how it performs its backpropagation process. Although we modified the overall structure of the model, the mathematical essence of the loss function remains the same as the one defined in the reference OpenAI paper.

### 3.2. Lazy Loading

It was previously mentioned that the size of the complete dataset was a critical factor that needed to be taken into account at some point.

By implementing a *lazy loading* interface utilizing the **TensorFlow dataset class** and exploiting the *map/reduce paradigm* with *lambda functions* we were able to load the actual pixel data of images from their image path only when needed. We wrapped this operation inside a custom function called `create_dataset` that exposes a general interface which allows for parametrization of the dataset creation procedure through parameters that influence *batching*, *shuffling*, *dataset structure*, *caching* and other aspects.

At this point we noticed that the vectorization for the captions was causing a minor *data leakage* from the test set to the train set, we corrected the mistake and implemented a new splitting function by wrapping the **scikit-learn train\_test\_split function** inside a custom function to also manage validation splits.

### 3.3. Pre-Trained Models

As previously explained, our implementation of the model was completely from scratch. At this point we encountered a performance bottleneck, impeding any further enhancement of the model. This was probably due to the huge amount of parameters in the model, which were quite a challenging task to train from scratch, even considering our large dataset.

As shown during the course lectures, we decided to try to leverage a *pre-trained* model for both our image and text encoder and perform *transfer learning*.

#### 3.3.1 Image Encoder

For the image encoder, we chose to utilize a **ConvNeXt** network via the **Keras Applications API**, specifically the **ConvNeXtTiny** variant, known for its compact size yet commendable performance. We made this selection based on two primary reasons: firstly, to opt for a reasonably sized and efficient CNN to mitigate the risk of overfitting, and secondly, due to our past projects demonstrating impressive results with this network, making it a logical starting point.

Further in the project development, we tried most of the CNNs available on the **Keras Applications API**, and achieved a substantial improvement using the **EfficientNetV2S** over the *ConvNeXtTiny*.

### 3.3.2 Text Encoder

As for the text encoder, we employed **BERT** as our preferred *text transformer*. We initially utilized the base version of the transformer as a foundation.

Importing and integrating the model as a layer within our architecture was challenging, given that most methods are designed for standalone use of these models for common tasks like classification. In the end we successfully overcame this obstacle thanks to the **tensorflow\_hub** module, importing BERT as a **KerasLayer** and incorporating it into our model. This allowed us to employ slimmed down versions of BERT to avoid overfitting, such as BERT 4L (BERT with only 4 transformer layers) and BERT 2L (2 transformer layers).

### 3.4. Embedding Projection

As previously explained, our architecture comprises the following components:

- an *image encoder* followed by a 2-layer dense neural network
- a *text encoder* followed by a 2-layer dense neural network

The purpose of the 2-layer dense network is to project the *high-level features* extracted from the encoders into a shared embedding space. In an effort to enhance this projection, we engaged in thorough experimentation. After extensive tinkering, we settled on a 2-layer dense network with *dropout* incorporated in the middle, along with a *residual connection* linking the two dense layers, ending in **layer normalization**.

We also explored various activation functions, starting with **ReLU** and subsequently testing **GELU**, **SiLU**, and **SELU**. Ultimately, SELU demonstrated a significant improvement, prompting us to adopt it for the projection.

Both dense layers were configured with 128 neurons, resulting in a 128-dimensional embedding space.

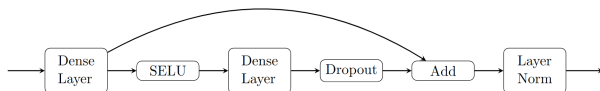


Figure 3: *Model Layout*

We experimented with various embedding space dimensions, initially reducing it to 64 and then further down to 32 after some tinkering. This 32-dimensional setting proved to be the best for a while. We also tried 16 dimensions but this configuration was not on par with the previous. It was only after significant modifications to our model and the use of *Data Augmentation* that we increased it back to 64, as detailed further in the report.

### 3.5. Loss Function Tweaks

We made several attempts to refine the loss function while implementing various changes to the model in hopes of enhancing its performance.

#### 3.5.1 Contrastive Penalization

Initially, we aimed to penalize the loss associated with similarities among distinct samples. This involved amplifying the loss tied to these similarities by a specified hyperparameter to further encourage the model to minimize similarities between different samples. Unfortunately, this approach proved unsuccessful, significantly compromising the model's performance.

#### 3.5.2 Weighted Loss

Our subsequent attempt involved employing a weighted average strategy. In the original CLIP paper, the authors independently computed the loss for the image encoder and the text encoder, then averaging these values. We attempted to improve this approach by weighting this average as follows:  $\alpha \cdot L_{img} + (1 - \alpha) \cdot L_{txt}$  (where  $L_{img}$  is the loss associated to the image encoder,  $L_{txt}$  to the text encoder). We experimented with both using  $\alpha$  as a hyperparameter and as a trainable parameter (confined between 0 and 1). However, both attempts yielded no improvement. Interestingly, in the case of the trainable parameter, the resulting  $\alpha$  value converged remarkably close to 0.5 after training. This indicated that employing a weighted average was futile, and a regular average remained the optimal choice.

#### 3.5.3 Geometric Mean

Given that weighting the mean did not yield improvements, we experimented with various types of means for computing the final loss value, primarily focusing on the *geometric mean*, represented in our case as:  $\sqrt[2]{L_{img} \cdot L_{txt}}$ . However, this attempt also proved unsuccessful, prompting us to retain the original formulation of the loss.

### 3.6. Encoder Pre-training

To enhance the stability of this intensive parameter training, we attempted to introduce a pre-training phase preceding the actual model training. The primary objective was to establish a more stable training process.

The approach involved pre-training the two encoders independently and without projections by incorporating a dense layer on top of them and training them in a multi-label classification scenario, making use of the provided concept dataset as labels. Subsequently, we would remove the dense classification layer and integrate the encoders into our CLIP model, followed by initiating the primary training phase.



Unfortunately, pre-training the text encoder did not yield any improvements; it only prolonged the entire training procedure. However, pre-training the image encoder resulted in a slight but consistent enhancement. Consequently, we opted to retain this initial pre-training for all subsequent iterations of the model.

### 3.7. Data Augmentation

To further improve performance we decided to insert data augmentation techniques inside our training and evaluation pipelines. All the augmentation modules are given a seed for random number generation in order to improve the reproducibility of the process.

#### 3.7.1 Classic Data Augmentation

Classic data augmentation (DAug) consists of adding minor perturbations of various natures on input training data in order to incentivize the model to generalize. Due to the modular nature of our training pipeline, to implement the DAug process, it was sufficient to insert a map call in the dataset creation function, utilizing a dedicated parameter containing the DAug pipeline.

We experimented with various different DAug pipelines, but the following gave us best results:

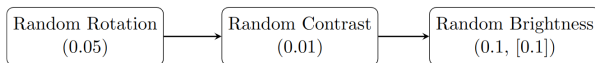


Figure 4: *Image Pipeline*

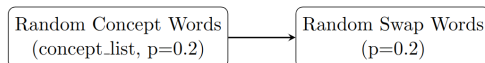


Figure 5: *Text Pipeline*

All the image augmentation modules are provided by the [Keras Layers](#) library and their parameters generally correspond to symmetric transformation ranges (except for the Random Brightness layer, where the first parameter is the factor range and the second parameter is the value range).

The text augmentation modules were implemented by us, drawing inspiration from the NLP Albumentations notebook by Alex Shonenkov [4]. The `RandomSwapWords` module takes a probability  $p$  as parameter, and with that probability creates a boolean mask for the caption associated with a certain input sample. The words highlighted by the mask are then randomly shuffled. The `RandomConceptWords` module takes a list of concept classes and a probability  $p$  as parameters, selects a random number of concepts based on  $p$  and the concepts of the input sample, randomly selects a position inside the caption

for each selected concept, and inserts the concept reference texts inside the caption at the chosen positions.

Even though the direct increase in performance caused by DAug was marginal, it opened up other paths for improvement that were previously unfeasible due to the model overfitting, such as using a larger BERT architectures and employing a bigger embedding size.

#### 3.7.2 Test Time Augmentation

Test Time Augmentation (TTA) is slightly different from DAug, as it is performed inside the model, at inference time. In order to reduce variance of predictions, the model is given `ttan` different augmented versions of the same input sample (plus the original inputs), the model is asked to perform a prediction for all of them, and then average the results to obtain a final prediction.

For TTA we utilized the same augmentation pipeline as in the previous step, with slightly modified parameters to make it more aggressive and without the `RandomConceptWords` module. We then implemented a wrapper for the encoders of the model, that takes as input the actual encoder of the model, an augmentation pipeline, and other TTA parameters (such as `ttan`). The wrapper produces a modified encoder that performs TTA with the desired parameters whenever is called.

Unfortunately there were no direct improvements by using TTA, at best it resulting in a performance equivalent to the base model. With TTA being a self-ensembling method, our guess is that the model is too stable on its predictions to gain any meaningful benefit from this technique.

### 3.8. Grayscale Preprocess

We noticed that most of the images contained in the dataset are grayscale images. However, they are stored as RGB images, with a shape of  $128 \times 128 \times 3$ . Being grayscale, the third dimension is completely redundant, and we know that additional dimensions can severely impact the model's performance. To address this, we implemented a solution by introducing a [Lambda layer](#) at the beginning of the image encoder. This layer calculates the average across the RGB channels (the third dimension) for inputs with three dimensions, effectively creating a simple color-to-grayscale conversion algorithm that outputs a 2-dimensional image. For inputs already in grayscale with two dimensions, the layer remains inactive.

This modification ensures seamless compatibility of the encoder with our existing infrastructure, requiring no alterations. Consequently, our model can effectively process both grayscale and RGB inputs. This way, our image encoder now operates on dimensions of  $128 \times 128$ .

The integration of this solution had a significant positive impact on the model's performance, resulting in substantial

improvement.

### 3.9. Evaluation

In the first steps of this project we were using the *loss* returned by the model as its quality metric, which is not really optimal for various reasons, therefore we decided to write a separate notebook solely dedicated to model evaluation for the two main retrieval tasks. An additional similar notebook was created with the purpose of comparing multiple models' performance on the established evaluation metrics.

The inference procedure is specular for image and text retrieval: we first generate the embeddings of the feature that we want to retrieve by calling the `predict` method of the relevant encoder on the dataset that we want to utilize. Then, for every query of the dataset we generate the embedding through the other encoder, and we compute the *cosine similarity* to then retrieve the *top-k* results for each query.

Finally, we compute the mean **accuracy**, **precision**, **recall** and **F1 score** for the query dataset through *micro-averaging*, and we visualize them utilizing *graphs* and *reports* with the possibility of comparing evaluations between models or against a random classifier baseline. However, we noticed there are multiple **relevance metrics** that are interesting to take into consideration (for simplicity, only image retrieval is illustrated):

- **Caption Equality:** a result is relevant only if its caption is equal to the original query; it can be quite restrictive since most of the captions are unique.
- **Concept Overlap:** a result is relevant only if it shares  $n$  concepts with the original element; it provides a good estimate for real use-cases but can be misleading if the threshold is too low, in addition it cannot be used if the concept data is not present.

Some other evaluation approaches were tried for the image retrieval scenario, such as the *Concept-Fusion Captions* approach, which consists of evaluating the model on samples where each caption was generated by concatenating the concepts of the corresponding sample. This method of evaluation steadily gave worse results than other approaches.

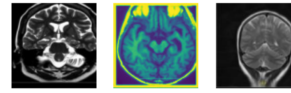
## 4. Results

### 4.1. Image and Text Retrieval Samples

Some sample outputs from the image and text retrieval tasks using  $k = 3$ .

For visualization reasons, both image and text data is shown in the results of both retrieval tasks; only the target feature is actually retrieved by the model in each task, while the other is obtained through a lookup of the dataset.

Top matches for query: "brain"  
 1) weight mri show right hippocampal volume loss and a mild dilation of the right temporal horn  
 2) axial contrast image show almost complete resolution of the temporal lobe lesion with mild dilatation of the left temporal horn  
 3) coronal flair on magnetic resonance imaging mri of brain

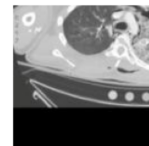


Top matches for query: "lungs"  
 1) man year old with right lung abscess after admission to the centre  
 2) the confirmation of right reexpansion by chest xray reexpansion pulmonary edema be also see  
 3) figure the double bubble sign



Figure 6: Image Retrieval

Top matches for query:



1) computed tomography take after bronchoscopy  
 2) preoperative chest compute tomography ct preoperative chest ct scan show a foreign body in the carina and right proximal main bronchus  
 3) ct scan demonstrate the resolution of infection it be well evident the chest drain within mediastinum on the right side that allow the drainage of infection

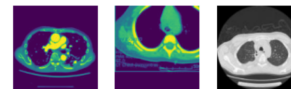


Figure 7: Text Retrieval 1

Top matches for query:



1) axial cone beam compute tomography view show two distinct canal in the mesiobuccal root of the maxillary right first molar tooth  
 2) determine the cbct cross section in an axial view  
 3) cbct show the border of the lesion and the perforation area

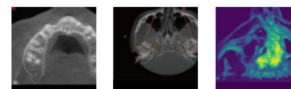


Figure 8: Text Retrieval 2

### 4.2. Model Architecture Improvement

The final iteration of the model emerged after implementing various tweaks and adjustments to its architecture and training methods mentioned in the previous section. However, not all of them resulted in an improved model. Some changes led to worse outcomes, while others showed no discernible impact. Each adjustment underwent meticulous evaluation using the methodologies mentioned before

(*caption equality* and *concept overlap*). Given the numerous changes to the model and the many more evaluation results for each, we have summarized the overall enhancement of the model through a corresponding chart.

This chart displays the **concept overlap accuracy** at  $k = 10$  and with a concept threshold of  $n = 2$  for each model modification, as it serves as a concise and clear metric to show the progression. Blue bars on the chart signify changes that yielded improvements in the architecture and were consequently integrated, while gray bars indicate changes that failed to enhance the model and were thus discarded.

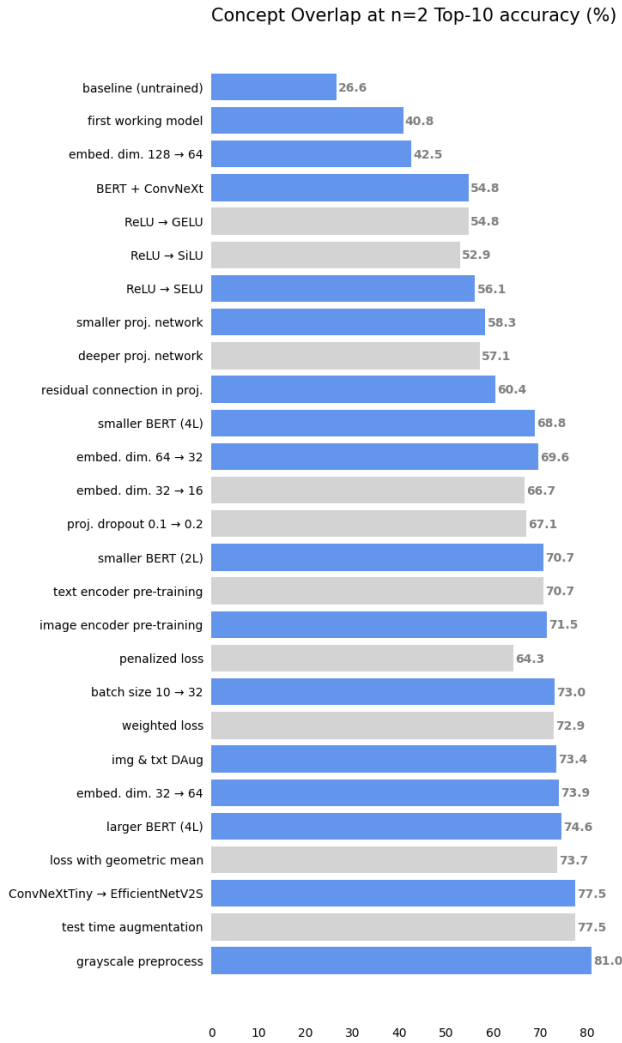


Figure 9: Model improvement overview

### 4.3. Embedding Space

To effectively demonstrate the model’s capabilities and the coherence of its *embedding space*, our aim now is to visualize its inner workings, specifically how the *visual en-*

*coder* and the *text encoder* interact. The primary goal of the model is centered on training both encoders to encode images and their corresponding descriptions within a unified embedding space. This process generates two latent vectors (one for images and one for captions) that ideally exhibit high similarity to each other while remaining distinct from other sample pairs within the set.

To illustrate this concept, we selected a random batch of 32 samples from the test dataset. We then employed their respective encoders to encode both the images and their associated captions. Subsequently, we normalized these embeddings and computed similarities by performing matrix multiplication between them. This process allows us to quantify the relationships and similarities between the encoded representations of images and captions within this shared embedding space on data that the model has never seen.

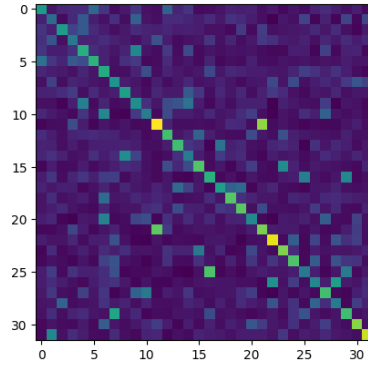


Figure 10: Diagonal trend in embeddings

As expected, our observations reveal a clear *diagonal trend* within the visualization. This trend distinctly illustrates that images and captions originating from the same sample show a notably high degree of similarity, denoted by a brighter color in the visualization. Conversely, when comparing images and captions from different samples, the similarity appears significantly lower, depicted by a darker color.

## 5. Zero-Shot Image Classification

Once our model had successfully achieved a high level of performance, we decided to put it to the test in a *zero-shot classification* scenario, as demonstrated and utilized in the CLIP paper.

The fundamental idea behind this approach involves having a set of textual classes serving as our classification labels. When presented with an image for classification, we encode both the image and the classes. Following this, we calculate the similarity between the embedding of the image and the embeddings of the classes. By applying a *softmax function* in order to sharpen the distribution, we determine

the highest similarity score, which becomes our prediction.

### 5.1. Custom Dataset

In order to comprehensively assess our model’s classification capabilities, we opted to construct a tailored dataset specifically designed for this purpose. Our rationale came from the observation that nearly every entry in the dataset caption explicitly delineated the type of analysis depicted in the corresponding image, such as ‘*x-ray scan of...*’ or ‘*ultrasound image of...*’. Hence, we decided to create a dataset where each sample comprised an image paired with a label representing the specific type of analysis conducted in that image.

To create this dataset, we utilized our test dataset, to avoid any potential biases. We built a piece of code to extract words from a predefined set of labels found within the image captions. These labels were subsequently employed as classes, allowing us to compile a custom dataset. The chosen classes are: **angiography, echocardiography, ultrasound, tomography, x-ray**.

Upon extracting the images from the test set, we ended up with a dataset comprising 4111 samples, with each sample associated with one of the specified five labels, with the following class distribution of samples: angiography: 299, echocardiography: 211, ultrasound: 520, tomography: 1873, xray: 1208.

### 5.2. Results

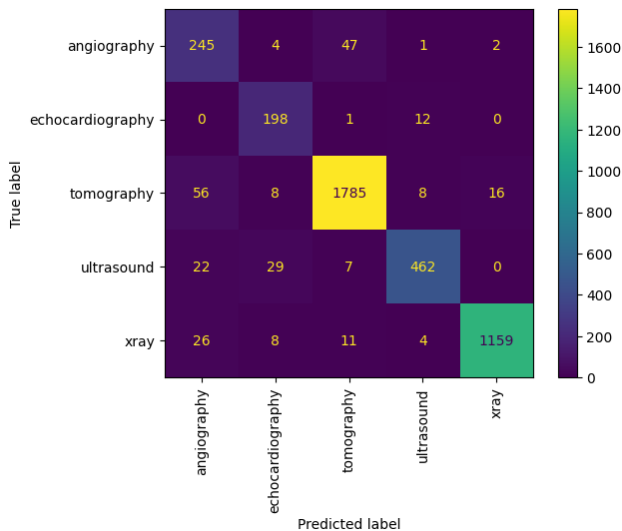


Figure 11: *Classification Confusion Matrix*

We then wrapped the classification procedure in a function and performed evaluation just as we would normally do with a trained classifier.

As it’s possible to observe from the confusion matrix visualization, the model’s predictions seem well-balanced as

a faint diagonal is present. However, there might be some slight bias towards the *angiography class* as it is the one with the most false positive predictions associated overall. This is possibly due to the conceptual overlap between the angiography technique and other types of exams such as x-ray and tomography.

These results led to an overall **accuracy** of **94%**, exceeding our brightest expectations.

## 6. Conclusions

In conclusion, this project faced several challenges, mainly due to the model’s unconventional architecture. Initially, we built both encoders from scratch to test the limits of this base design and apply our knowledge about the *transformer* architecture that we learned from the course. However, this approach started showing limitations over time, leading us to shift to pre-trained models.

To improve the model, we used traditional deep learning techniques such as adding *skip connections*, data preprocessing to reduce dimensionality, data augmentation, alongside more unconventional and creative methods. Overcoming overfitting was one of our major challenges, which we addressed by reducing model parameters, employing smaller encoders and projections, and using data augmentation. Moreover, dealing with a large model and a huge dataset required optimizations like *lazy loading* to work effectively with the complete dataset.

Our efforts to adjust the loss function proved unsuccessful. This demonstrates that the base CLIP loss function implemented by OpenAI performs exceptionally well with this architecture, leaving little room for improvement, even in our specific scenario.

Assessing the model’s performance was challenging due to its unique architecture not geared towards common tasks like regression or classification. To address this, we developed metrics like *Caption Equality* and *Concept Overlap* for detailed evaluation. Visual inspection and plotting of computed similarities consistently showed a diagonal trend, confirming its functionality. Furthermore, the remarkable results in zero-shot classification undeniably demonstrated the model’s effectiveness and generalization capabilities.

## References

- [1] O. Pelka, S. Koitka, J. Rückert, F. Nensa, and C. M. Friedrich. Radiology objects in context (roco): A multimodal image dataset. In D. Stoyanov, Z. Taylor, S. Balocco, R. Sznitman, A. Martel, L. Maier-Hein, L. Duong, G. Zahnd, S. Demirci, S. Albarqouni, S.-L. Lee, S. Moriconi, V. Cheplygina, D. Mateus, E. Trucco, E. Granger, and P. Jannin, editors, *Intravascular Imaging and Computer Assisted Stenting and Large-Scale Annotation of Biomedical Data and Expert Label Synthesis*, pages 180–189, Cham, 2018. Springer International Publishing. 1



- [2] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021. 1
- [3] A. Radford, I. Sutskever, J. W. Kim, G. Krueger, and S. Agarwal. Clip: Connecting text and images, 2021. 1
- [4] A. Shonenkov. Nlp albumentations, 2019. 5