

# LUS Images classification with uncertainty detection and image similarity

Davide Modolo (#229297)

## Abstract

The proposed multi-stage model for predicting LUS image scores is built using three main components: a multi-class frame classifier, an uncertainty detection model, and a similarity module. The idea is to retrieve similar images and analyze them when the initial prediction is uncertain. The entire project is currently available on GitHub [1].

## Introduction

This project aimed to develop an alternative way to predict LUS images scores. The first idea that came to my mind was to build something that could be used by doctors, retrieving similar images when the score of a specific frame was not sure to “help” with the decision.

The existing methodology consists in scoring all 14 different spots and summing their values. If the result is  $< 24/42$ , the patient can be left going home because it indicates a low probability of worsening.

As explained in the article by S. Roy et al. [2], LUS images are scored as:

- 0: no artifact in the picture;
- 1: at least one vertical artifact (B-line);
- 2: small consolidation below the pleural surface;
- 3: wider hyperechogenic area ( $< 50\%$ ) below the pleural surface.

Frames are from videos taken using ultrasound probes in a maximum of 14 different spots (6 on the front and 8 on the back of the patient), as explained in the article by G. Soldati et al. [3].

My proposed model consists of three main components:

- A multi-class frame classifier that predicts the score of individual LUS images;
- An uncertainty detection model that evaluates the confidence of the initial prediction;
- A similarity module that retrieves similar images and analyzes them when the first model is not confident.

With this multi-stage approach, I aim to provide technicians with a different (hopefully reliable) diagnosis tool.

Unfortunately, we will see this idea is probably not effective and the results are not encouraging.

## 1. Data

We have been given a partial dataset from the San Matteo hospital, consisting of 11 patients for a total of  $\sim 47k$  frames.

The dataset score distribution is shown in Figure 1a; at first glance, it could seem to be almost balanced (with only the score 1 that has fewer frames), but in reality, many patients are inherently unbalanced (the score distribution for each patient is shown in Figure 2).

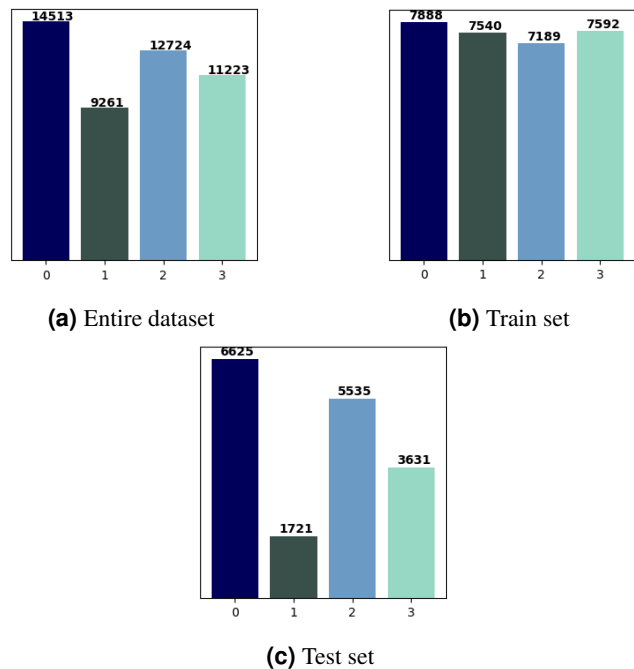


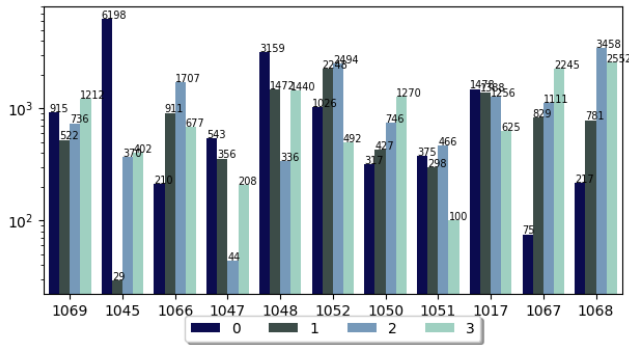
Figure 1. Score distribution in the dataset

### 1.1 Augmentation

Using the raw dataset got me overfitting even during the first epoch. To address this issue I implemented some transformations taken from the article [2].

In specific, each transformation is activated with a probability of 50%. The set of my augmentation function is:

- affine transformations (translation =  $\pm 15\%$ , rotation =  $\pm 15^\circ$ , scaling  $\pm 45\%$ , and shearing =  $\pm 4.5^\circ$ );
- multiplication with a constant ( $\pm 45\%$ );



**Figure 2.** Number of frames for each score for each patient (log scale for better visualization).

- Gaussian blurring ( $\sigma = 3/4$ );
- horizontal flipping ( $p = 0.5$ ).

## 1.2 Data splitting

Having 11 patients available, my idea was to use 8 of them to train the model and the remaining 3 for testing. This was due to the fact that using a portion of the frames for a patient in the test set and another in the training set easily leads to overfitting. Even dividing by exams would not be effective since different exams for the same patients still have a big correlation.

The first attempt was to test with a  $k$ -fold approach and then choose the best configuration, but having 165 combinations with  $> 4h$  of computing time per combination was unfeasible.

So, to balance the dataset I computed the standard deviation within scores for each 8-patient combination and selected the one with the lowest std (Figure 3), resulting in the division shown in Figure 1b; the problem now was with the test set, that resulted to be very unbalanced (Figure 1c). After different attempts to balance both sets, I decided to just select an equal number of images for each score from the testing patients set of frames to use in the `test_model` method (still, confusion matrices on this report are built using the entire available test set).

## 2. Multi-class classifiers

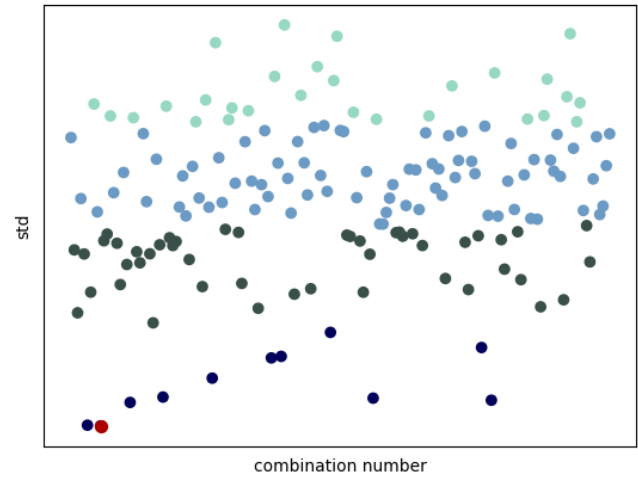
The first module of my project consists in a deep learning classifier that predicts the score from a frame.

Different pre-trained models have been tested with several different values for my hyperparameters. The training part has been made several times in order to find a model that didn't overfit in the first epoch or didn't output only one single score.

In general, using models that are too big could lead to overfitting and using models that are too small could get no good generalization capability.

### 2.1 ResNet18

ResNet (Residual Network) is a network introduced by K. He et al. [4] trained on the ImageNet dataset [5].



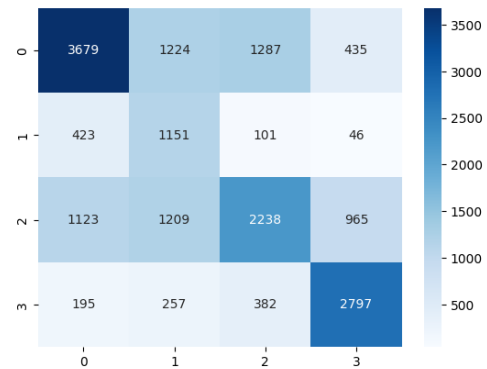
**Figure 3.** Standard deviation within the number of frames per score of every combination of 8 patients, the red one is the minimum (and so, it is the selected combination).

There are different versions of this model based on the number of layers. Looking for a “small” model, ResNet18 was the smallest one and so it has been selected for testing.

After many runs, I was able to achieve an accuracy of  $\sim 56.65\%$  in my test set before overfitting. The confusion matrix on the test set can be seen in Figure 4, resulting in an accuracy class-wise that can be seen in Table 1.

0	1	2	3
55.53%	66.88%	40.43%	77.03%

**Table 1.** Accuracy class-wise of the fine-tuned ResNet18.



**Figure 4.** Confusion matrix of the fine-tuned ResNet18.

### 2.2 VGG16

VGG (Visual Geometry Group) is a convolutional Neural Network built by K. Simonyan, A. Zisserman [6]. It has been trained on a subset of the ImageNet dataset.

Similarly to ResNet, VGG is available with 16 and 18 layers. For the same reasons as above, VGG16 has been selected and tested.

Independently from my fine-tuning tries, VGG16 started memorizing the training data in the first epoch (even having fewer parameters than ResNet18).

### 2.3 SqueezeNet

SqueezeNet is a model developed by F. N. Iandola et al. [7] in 2016 trained on ImageNet.

Following the idea to find a compact model, I found this variation of AlexNet that is still capable of very good performance while requiring fewer parameters.

SqueezeNet gave me the best results in the early stage of the project, but after refining the fine-tuning of the ResNet, I decided to not use it.

### 2.4 Built-from-scratch model

I even tried building from scratch a Convolutional Neural Network. I tried different combinations of Convolutional layers but the results were very poor, resulting in a path I didn't follow deeper.

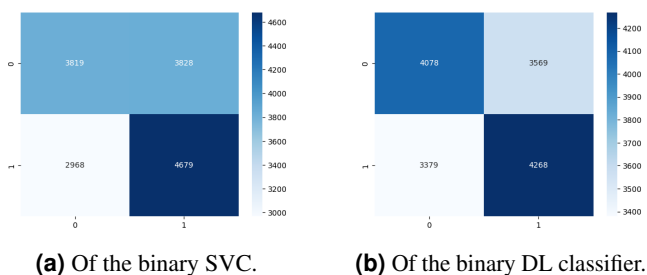
## 3. Binary classifiers

The second goal of this project was to develop a mechanism to determine the confidence of the multi-class frame classifier in its predictions.

Initially, I explored the possibility of using a threshold-based approach; however, during the in-class presentation, we realized that a more sophisticated approach would be interesting to try to capture the behaviour of the model in both correct and incorrect predictions. In addition, I observed that the maximum softmax values were very similar, indicating that a simple threshold approach would not be effective.

As a result, I developed two binary classification models that use the softmax values of the first model to evaluate the confidence in its predictions.

To train these models, the data has been built by using the trained ResNet18 model on the training set to then save the output values with their correctness (T/F); I also had to balance the new dataset since even a bit of unbalance in it would lead to having one and only one output value.



**Figure 5.** Confusion matrices of binary classifiers.

### 3.1 Deep model

For this approach, I built a simple neural network with four inputs and two outputs. It uses one dense layer with a Sigmoid

activation function to make predictions (ReLU has also been tested but resulted in worse performance).

Any layer or complexity I added resulted in a very bad performance, with this configuration I was able to achieve the 54.66% in one of many runs (Figure 5b).

### 3.2 Support Vector Classifier

From the `sci-kit learn` library I took a linear model that tries to find the best hyperplane that separates the two classes. The SVC model can be trained quickly and its performance was similar to the DL model I created, but I found it more prone to “overfitting” since even a dataset that was a little unbalanced in the class distribution resulted in 50% (so random choice). The final SVC got 56.34% accuracy (Figure 5a).

### 3.3 Four SVCs

To investigate further the behaviour of the SVC, I decided also to test one different SVC for each class with the idea of: “Depending on the class predicted by the first classifier, I call one of the four SVCs trying to have a more specific approach”.

The average accuracy by class is 54.2%, worse than the single SVC. The results seemed fine until I tried it in the final model where it got worse results than both the Deep Learning model and the single SVC.

## 4. Image similarity

The third module of my project consists of an image similarity model. As briefly said before, this idea was born from the definition of diagnosis that has some degree of subjectivity. Of course, thanks to the scoring mechanism proposed and cited in Introduction, this effect is mitigated. Still, I was interested in showing similar images when the score prediction had low confidence.

Only a portion of the available images from the train set has been used since every approach has a working time too high with the entire dataset. I decided to randomly pick the same number of images for each score for each patient from the training set (1408) and again in the test set (348) to compare every approach (but of course having fewer patients, results are not very reliable).

### 4.1 Near Duplicate Image Search

Near Duplicate Image Search is a technique for finding similar images by using a nearest neighbour algorithm. The `annoy` library is a fast and efficient approximate nearest neighbour search library.

The process involves converting each image into a high-dimensional feature vector, I used the ResNet18 features. The feature vector is then indexed using the Annoy library, which allows for retrieval of the most similar images based on their Euclidean distance.

This approach has been discarded since its performance seemed good, getting a maximum accuracy of 59.06%, in-

vestigating the behaviour, I found out that almost 60% of the images haven't any nearest neighbour, so, even if the final model with this component has the highest numerical accuracy, it does not follow the idea of finding similar images when required. This issue could probably be resolved by increasing the number of images selected to build the feature vector, but this process would be very slow. In addition, for the images with available nearest neighbours, the only score that was output was 2.

## 4.2 t-SNE

t-SNE (t-Distributed Stochastic Neighbor Embedding) is a machine learning algorithm used for data visualization. It maps high-dimensional data points into a low-dimensional space, typically two or three dimensions while preserving the pairwise similarities between the points.

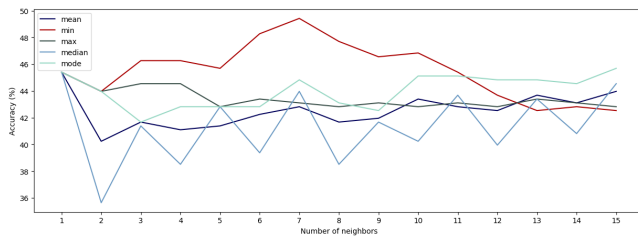
It's a technique used in computer vision to find images that are visually similar to a given query image. The goal is to identify images that are duplicates, or nearly identical, to the query image.

While both t-SNE and near duplicate image search are used in computer vision, they have different purposes and applications. t-SNE is used for visualizing high-dimensional data, while near duplicate image search is used specifically for identifying visually similar images.

So, while Near duplicate image search should fit better the scope of this project, t-SNE got me better results overall.

### 4.2.1 Embeddings

Using the fine-tuned ResNet18, I extrapolated the embeddings from the entire training set. I then built a t-SNE for each test image and computed the mean, minimum, maximum, mode, and median values across the scores of 1 to 15 neighbours. The best result I got was 49.42% accuracy with 7 neighbours and the function *minimum* (Figure 6).



**Figure 6.** Comparison between neighbours in t-SNE using ResNet18 embeddings.

**Embedding with no t-SNE** While working on the t-SNE based on embeddings, I decided to try using the embeddings as they were, using the cosine similarity to find the closest  $n$  images (Euclidian distance has also been tested but resulted in even worse results).

“It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction” [8].

It got better results than Near Duplicate Image Search (since it was able to output different values) but worse than any t-SNE approach, with an accuracy on the entire test dataset of 38.46%.

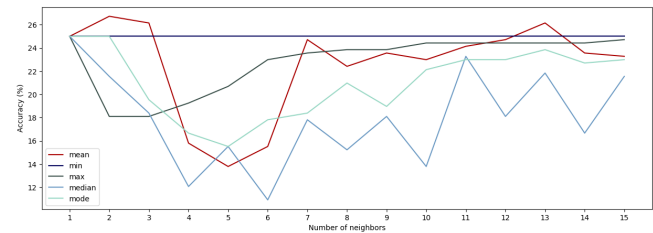
This approach was tested and the best configuration was found by taking the *mode* of the scores across the 14 nearest images.

### 4.2.2 Raw Images

I also tried using raw images to create a t-SNE map, flattening each image and treating it as a 1D vector.

As in the Embedding try, I computed the mean, minimum, maximum, mode, and median values across the scores of 1 to 15 neighbours.

In this case, results were not promising since the maximum accuracy was 26.72% with 2 neighbours and the function *mean* (Figure 7).

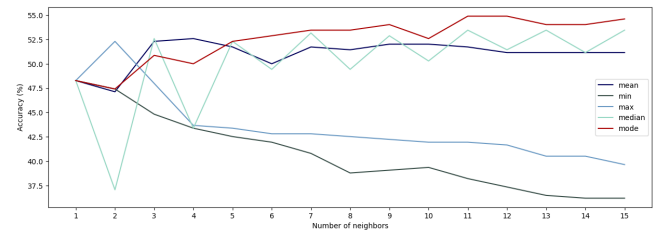


**Figure 7.** Comparison between neighbours in t-SNE using raw images as inputs.

### 4.2.3 Behavior

Last, I tried to build a t-SNE using the softmax values of the first classifier. Even if this representation is not based on visually similar images, it got me the best results across all the t-SNE approaches.

As before, I computed the mean, minimum, maximum, mode, and median values across the scores of 1 to 15 neighbours. The best accuracy was 54.88%, found with 11 neighbours and the function *mode* (Figure 8).



**Figure 8.** Comparison between neighbours in t-SNE using classifier softmax results as inputs.

## 5. Performance analysis

Regarding the final model, I tested almost all possible configurations of the three components. Although the behaviour of each set of three modules was quite similar, the configuration

that achieved the best accuracy was the one consisting of the three models that individually produced the best results.

The three components used in the final model are:

- ResNet18
- Single SVC
- t-SNE behaviour version

I decided to run again all the test shit with the same subset of images in every module. Since t-SNE is very slow, the result testing set is made of 348 images. The confusion matrices for this new test set of each module and the final model are in Figure 9.

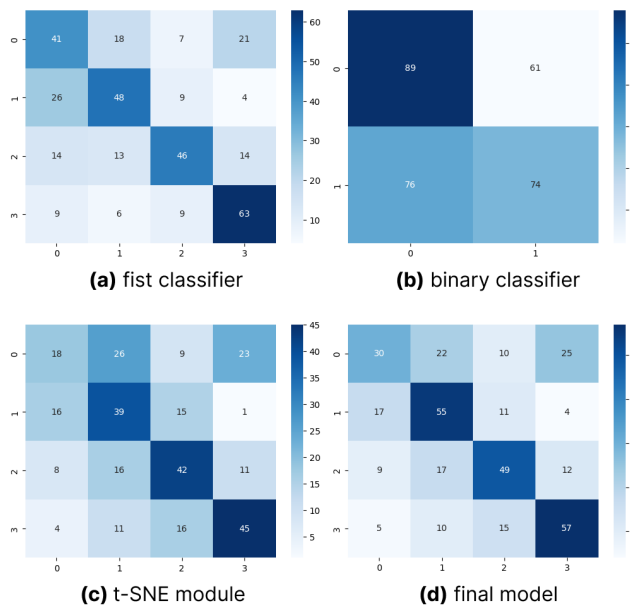


Figure 9. Confusion matrices

Unfortunately, the average accuracy of the final model on this new test set is very similar to the one of just the classification model.

Starting from a 56.9% accuracy of the classifier, with the 54.33% of the binary and with 48% of the t-SNE module, the final model achieved a 54.89% accuracy in the end.

Having three components in the model means having three potential points of failure. In specific, I tried to analyze the behaviour of the binary model in the final one and it showed that, probably due to the little amount of data I used to train it, it only got the right prediction on 3059/17512 images (entire test set), getting almost always the correct result for the label 1, less than 25% of the time with label 2 and very few times with labels 0 and 3.

In addition to this, the similarity module uses only a little subset of the training images and it could be that most of the time the binary model states the first one is not a correct prediction it overlaps with the wrong results from the analysis of the similar images.

In general, I'm not completely able to find one component that breaks the entire machine; looking at the confusion matrix,

we can see that performances are similar across all the scores and all the modules.

## 6. Conclusions

Based on the analysis, the proposed model did not achieve the desired results. Despite testing multiple configurations and using various components, the final model's accuracy was slightly worse than the one of the classification model alone. Furthermore, one of the modules (the binary model) did not perform well, potentially due to the limited amount of data used to train it.

It is also possible that the similarity module's reliance on a small subset of training images could have led to incorrect predictions. While I wasn't able to identify a single component responsible for the model's lack of success, it seems that having multiple potential points of failure could have contributed to the final performance.

Overall, it appears that the proposed model needs further improvements and adjustments to achieve better or more valuable results. However, it is important to note that the project was still valuable as it provided insights and learning experiences, but I fear my idea is not effective in a practical way despite my best efforts.

### 6.1 Future works

Since this project is built with many components, several different things can be done to try to improve it:

- Test more pre-trained models: there are dozens of models out there that are capable of multi-class image classification, there could be one that works better with this specific project;
- Use more data: As the binary model's poor performance may have been due to limited data, collecting more data could help improve the binary model's accuracy;
- Locality Sensitive Hashing: it's a variant of the Near Duplicate Image Search methodology that uses a special hashing function to encode the images, but the implementation is not trivial;
- Speed up t-SNE using the CUDA version developed by Chan et al. [9], in order to use more images for each t-SNE build or test different configurations of this method.

## References

- [1] GitHub repository with the project. [Online]. Available: <https://github.com/davidemodolo/Lung-Ultrasound-Image-Classifier>
- [2] S. Roy, W. Menapace, S. Oei, B. Luijten, E. Fini, C. Saltori, I. Huijben, N. Chennakeshava, F. Mento, A. Sentelli, E. Peschiera, R. Trevisan, G. Maschietto,

- E. Torri, R. Inchingolo, A. Smargiassi, G. Soldati, P. Rota, A. Passerini, R. J. G. van Sloun, E. Ricci, and L. Demi, "Deep learning for classification and localization of covid-19 markers in point-of-care lung ultrasound," *IEEE Transactions on Medical Imaging*, vol. 39, no. 8, pp. 2676–2687, 2020.
- [3] G. Soldati, A. Smargiassi, R. Inchingolo, D. Buonsenso, T. Perrone, D. F. Briganti, S. Perlini, E. Torri, A. Mariani, E. E. Mossolani, F. Tursi, F. Mento, and L. Demi, "Proposal for International Standardization of the Use of Lung Ultrasound for Patients With COVID-19: A Simple, Quantitative, Reproducible Method," *J Ultrasound Med*, vol. 39, no. 7, pp. 1413–1419, Jul 2020.
- [4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.
- [5] ImageNet site. [Online]. Available: <https://www.image-net.org/>
- [6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.
- [7] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," 2016.
- [8] Cosine Similarity. [Online]. Available: <https://www.sciencedirect.com/topics/computer-science/cosine-similarity>
- [9] D. M. Chan, R. Rao, F. Huang, and J. F. Canny, "Gpu accelerated t-distributed stochastic neighbour embedding," *Journal of Parallel and Distributed Computing*, vol. 131, pp. 1–13, 2019.