Left Ventricle Automatic Segmentation with Deep Learning on the EchoNet Dynamic Dataset

Omar Alejandro Rodríguez Valencia, Cristian Javier Cázares Molina, Jair Josué Jimarez García, Musel Emmanuel Tabares Pardo, and Siddhartha López Valenzuela

Tecnológico de Monterrey, 45138 Zapopan, Jalisco, México

Abstract. With the growth in Deep Learning techniques, it is only natural that this technologies are growing also among the medical area. In this paper, the researchers use U-Net to compare how mask-based and landmark-based left ventricle image segmentation behave on the EchoNet-Dynamic data set provided by Stanford University. After explaining how the data was extracted from the dataset, the authors found that the mask model showed better dice score results (0.98 for masks and 0.21 for landmark), but discovering and giving possible solutions to problems like unbalanced classes and an unreliable loss function election.

Keywords: Echonet-Dynamic · Image Segmentation · U-Net

1 Introduction

Machine Learning (ML) is a tool that allow us to predict values and find patterns given a data set. Because of the fast development this technology has experienced, it is only natural that many other areas intend to take advantage to predict phenomena and solve problems that may be too hard for a human, but are easy to compute. One of this areas is, of course, the medicine, that has been trying to use ML to predict which condition or disease a patient may suffer or, even find patterns to work and find why some diseases happen (Shemona & Chellappan, 2020). In this paper, we will make use of the EchoNet Dynamic dataset, which contains images taken from echocardiograms in order to train a Deep Learning model that can make an appropriate mask i.e., another image in which the left ventricle (the area of interest) is isolated from the rest of the image so that it can be appropriately recognized by other technologies in the future. This specific technique is called "Image Segmentation".

It is of interest that this problem is solved because it can be helpful for future medical investigation, from diagnosis, to predictions based on the information that doctors can extract from the heart's left ventricle.

1.1 Scope

The objective of this investigation is to assess the technique that shows better results generating accurate masks that identify correctly the heart's left ventricle given an echocardiogram image. The techniques used to extract the masks

and train the models will be explained in Section 3. The models (one based on landmarks and the other on masks) were be evaluated with a new dataset, which contains images with more noise than the one used for training.

Although the dataset contains more in-depth clinical measurements that could be used to predict some medical conditions or abnormal behaviors inside the heart (as explained in Section 2) that is outside of this paper's scope, so that data will be ignored.

1.2 Background

The use of Deep Learning with medical purposes has been widely spread in problems that require recognition, for example, trying to detect cancerous cells (Nasim et al., 2023). This investigations have been done with the objective of diagnosing different diseases faster and with a lesser cost.

On the other hand, the so called "Fully Convolutional Neural Networks" have been widely used in tasks that include image segmentation (Long, Shelhamer, & Darrell, 2015). These networks assign a class to each pixel in an image instead of labeling the image as a whole. Since image segmentation consists in the creation of masks that label if the fraction of an image is part of our area of interest, the technique can solve this problem by assigning the label "On" or "Off" pixel by pixel. "On", meaning that the pixel is part of our area of interest and "Off", meaning that it is not.

Investigations aiming to create a mask of the heart's left ventricle have been found, one example is the one done by Nast-Esfahani (2018), but that paper, while reporting acceptable results (87.24% dice score), used magnetic resonance images, not echocardiograms. Anyway, the research's good results suggest that our particular problem may be solved using a fully convolutional network, which is the technology that will be used further in this paper (Nasr-Esfahani et al., 2018).

2 Data

The data set provided by Stanford University consists of 10,030 echocardiography videos (which adds up approximately 7.9 GB) dated from 2016 to 2018 as part of routine clinical care. While this is a considerable amount of data, it is still possible to handle it without using Big Data tools. Each video has approximately 100 frames and it must be noted that the videos were cropped to remove text and information that is not part of the scanning itself, as show in Fig 1. The resulting images were resized into 112X112 pixels and some were paired with a mask that was used to train the model, as shown in Fig 2. Each image is paired with a segmentation mask, which is a binary image (taking only ones and zeroes as values) that can be applied to the original image in order to isolate the relevant object from the less important ones. This masks are given in form of coordinates that represent essential points to identify the left ventricle. (Ouyang, 2020)

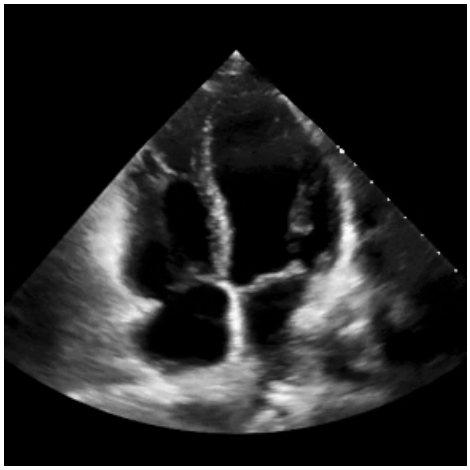

Fig. 1. Sample image from dataset. (Ouyang, 2020)

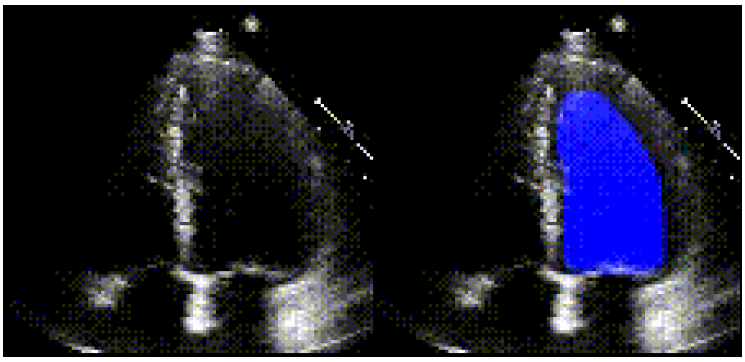

Fig. 2. Echocardiogram paired with its mask. (Ouyang, 2020)

The left ventricle is specially important to identify because it has the function of sending oxygenated blood for its distribution to the body, making any malfunction to result in heart failure, so its quick identification can be vitally important for diagnosis or prevention. (Berman, Tupper, & Bhardwaj, 2022)

In addition to the videos, the data set also contains clinical measurements like eject fraction, end systolic volume or the end diastolic volume. These measurements, however, are outside of this project's scope, since the problem that will be addressed in this paper is only to generate masks tracing the left ventricle so that they can be used in the future for any purpose that requires that specific part to be identified correctly, as explained in Section 1.1.

As the data set was provided by Stanford University, it is assumed that it follows laws concerning privacy on medical data. The way to access this data is by registering in the Stanford web page where the information can be found (which is also cited in this paper). Anyway, the data set was stored only locally, avoiding any repository outside of the one used for retrieval.

3 Methodology

In this section, the approaches to solve the problem and the architecture used will be explained in detail, as well as the software and hardware used for solving the problem.

3.1 System specifications

The computer in which the models were created is a laptop with 8 GB RAM and a 3070 Ti NVIDIA graphic card.

The Python version used was Python 3.11. To create the model, we will use PyTorch (Paszke et al., 2019), which is a framework created specifically for designing and implementing deep learning models in Python. The framework's version is 2.1.1. Other libraries used in the project will be listed next:

- requests==2.31.0
- setuptools==69.0.2
- numpy==1.26.2
- pandas==2.1.3
- matplotlib==3.8.2
- pillow==10.1.0
- torchvision==0.16.1
- opencv-python==4.8.1.78
- scikit-learn==1.3.2
- scikit-image==0.22.0
- seaborn==0.13.0
- ipython==8.18.1
- tqdm==4.66.1
- albumentations==1.3.1
- torchmetrics==1.2.0

3.2 Deep Learning

Deep Learning is a technique that works similarly to a Linear Regression, to be more precise, its foundation is a perceptron, which takes any number of inputs and processes them by finding the weights that best fit the data given in order to find a model capable of predicting unseen data. The difference between a Deep Neural Network (DNN) and a simple perceptron is that a DNN has any number of layers containing any number of neurons each, which provide a different interpretation of the data and feature extraction. This architecture can fit more variety of data than other, simpler, techniques. (Alzubaidi et al., 2021)

Another important concept to understand how Deep Learning works is the loss function. A Deep Learning challenge can be visualized as an optimization problem, where the error needs to be minimized to ensure the best possible model. To do this, Deep Learning models use stochastic gradient descent (which is an iterative algorithm that moves through a function's slope) to find the approximation of the lowest value on a function that calculates the difference between the real values and the predicted values. That function is called "loss function". (Jadon, 2020)

3.3 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a technique within Deep Learning used for tasks that involve images, whether it involves classification, detection, etc.

What makes special a CNN are the convolutional layers, this layers focus on kernels, which are n-length filters that extract features inside an image in order to complete the task given by fitting the data used for training.

Another important layer in a CNN is the pooling layer, which is used to reduce the image's dimension to make the model simpler and, in consequence, faster. One type of pooling that exists and is used is the max pool, which, as shown in Fig 3, makes a pixel take the max value of the kernel as its new value. (O'Shea & Nash, 2015)

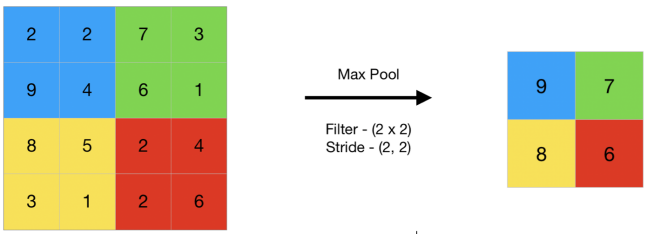

Fig. 3. Max pooling kernel.

3.4 Architecture used (U-Net)

To solve this problem, we will make use of the U-Net (Ronneberger, Fischer, & Brox, 2015) architecture, which was created in 2015 specifically to solve biomedical image processing tasks, so it is an architecture designed to solve problems akin to ours.

Some other reasons why the U-Net architecture is widely used for medical tasks is that it was designed to solve segmentation problems, it does not require a lot of data to be trained effectively and it has some characteristics (contracting and expansive path that will be further explained later on) that allow it to capture both global and local features.

The architecture, shown in Fig 4, is a fully convolutional network, i.e., it assigns a class to each pixel in an input image, and receives its name because of its U-like shape, which it takes because it consists of two paths: contracting and expansive (or encoder and decoder). The contracting path, follows the normal convolutional network architecture: a sequence of 3X3 convolutions, the application of ReLu functions and 2X2 max pooling with stride 2 for downsampling. It serves the function of feature extraction. The expansive path then upsamples the feature map to revert the changes that were made in the contracting path in order to increase the resolution of the output and generating the final mask.

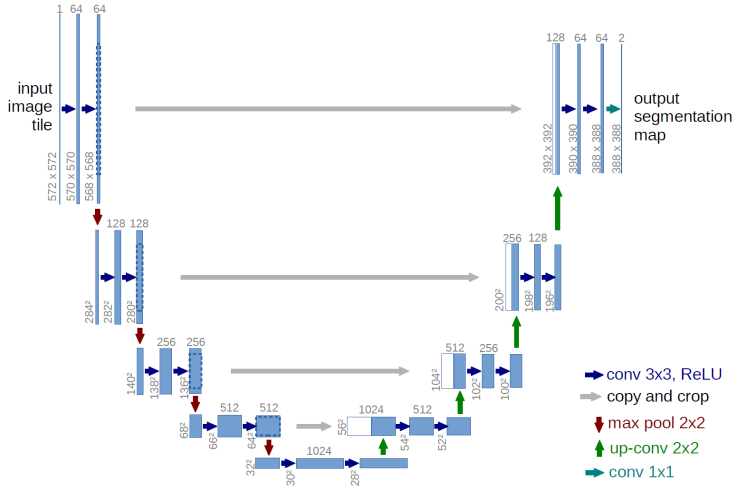

Fig. 4. U-Net architecture. (Ronneberger, Fischer, and Brox, 2015)

3.5 Mask approach

In this approach, the model will receive the original image as input and will return the whole mask as the output. The mask is an image that can only take 0's and 1's as values for each pixel.

Of course, to train the model we needed to create the masks using the coordinates mentioned in section 2 and, once they are extracted and paired with its specific frame, they were given to the model. The loss function used for this approach was the binary cross entropy, which is a function used to measure the difference between predicted and real binary labels and follows the next formula:

$$BCE = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

Where y is the real label and $\hat{y}$ is the predicted label. This can be done because image segmentation using a Fully Convolutional Network can be considered a pixel by pixel classification problem, where the two possible values are 0 and 1. (Jadon, 2020)

The validation metric used was the dice score, which follows the next formula:

$$Dice = \frac{2 * |X \cap Y|}{|X| + |Y|}$$

Since the two resulting images can only take 0's and 1's as values, the formula compares each pixel and finds how similar the masks are, taking a final result between 0 in the case that the masks are totally different and 1 if both images are exactly the same.

As explained in Section 2, the dataset does not contain the masks themselves, rather, it grants coordinates and creating the corresponding masks with those coordinates is left upon us, which is no trivial task and needed some work to be made in order to find the technique that would return the best masks.

Three techniques were tried to solve this problem, but all of them used the "fillPoly" function which can be found in OpenCV library (Bradski, 2000) that can be used to draw filled polygons taking three parameters as input: the image on which the polygon will be drawn, the polygon's points, which, in this case, were the coordinates given by the dataset and the color of the polygon.

The way this function is used is simple: first, we created a black image with the same dimensions of the original image, then, we called the "fillPoly" function receiving that image, the array containing the appropriate coordinates and the color white, represented in RGB manner.

The reason why some work needed to be done is that this function does not work well when called using the original coordinates, as they do not seem to follow the required order to get the appropriate polygon. So, the methods described next are, essentially, different ways to order the coordinates so that the mask obtained is the smoothest possible.

Simple Sort method: This method is the most simple of them all, a sorted array is created using the "sorted" Python function, which only takes an iterable as input and returns the same iterable, but sorted in an ascending or descending manner.

Centroid method: In this method, we first create a centroid, using the mean between all the points given by the dataset and, then, we use that centroid in the Numpy function "arctan2" to create a sorted array of coordinates which can be used with the "fillPoly" function.

Convex Hull method: In this method, the function used is OpenCV's "convexHull", which scans the array of coordinates using Sklansky's algorithm and sorts them, returning an array that can be used with the "fillPoly" function.

3.6 Landmark approach

In this approach, the model will receive the original image as input and will return a probability map showing each pixel's chance to be a landmark as output. This output will be presented in the form of a n-dimension tensor, where n is the number of landmarks predicted.

A landmark is an important point that can be used to identify the object of interest. The way landmarks are chosen is specific to each problem, but the main idea is that, given a set of landmarks, the overall shape of the object can be reconstructed.

The main conflict with this approach is the way the loss function can be handled, because there is no objectively "correct" landmark, even if one was selected by a human. So, if only the image and the coordinates are given to the algorithm, the model will consider those specific coordinates as the only correct option and will punish every other prediction in the backpropagation, even if the predicted landmark was really close to the one given as label. To avoid this, during the training, the model was given a probability map to compare and make the backpropagation correctly, punishing harshly only predictions that are far away from important points or borders in general, but will punish lightly or not at all predictions that are close to the label given. The loss function given to the model is the Cross Entropy, which follows the next formula:

$$CE = -\sum y \log(\hat{y})$$

Where y is the true value and $\hat{y}$ is the predicted value. Note that, when there are only two classes, this formula becomes the one described in Section 3.5. (Jadon, 2020)

Another matter to take into account while implementing this method is the amount of landmarks needed. The minimum number of landmarks is three (as two will only create a line as a mask and one is nonsense), marking the most important points that delimit the ventricle as a triangle, as shown in Fig 5. The problem with using only three landmarks is that the mask will not fit the image accurately when the time comes to link the landmarks and create the mask, as the result would be just a triangle.

On the other hand, the problem in using every coordinate given in the dataset (which sum around 20 per image) is that the final algorithm will try to predict the

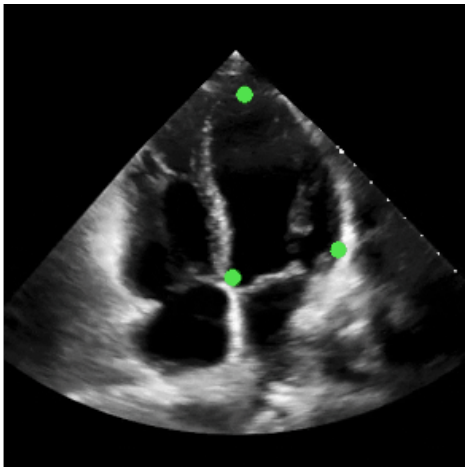

Fig. 5. Three landmarks overlapping echocardiogram.

same number of landmarks, increasing the model's number of learnable weights significantly, thus, slowing it down considerably.

To try to balance these two situations, a good number of landmarks may be seven, as shown in Fig 6. That way, we can not only mark the three most important points, but also check if there is an additional curvature in the ventricle without adding too much weight to the model, but the number of points taken is up to the researcher.

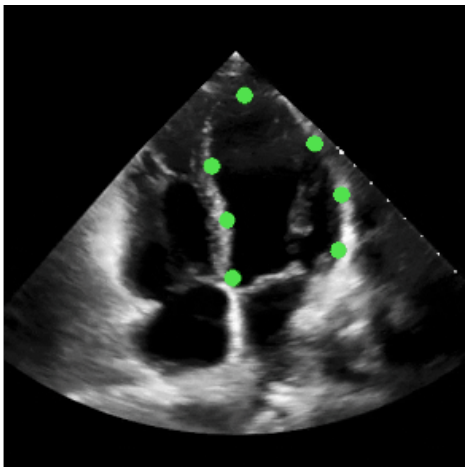

Fig. 6. Seven landmarks overlapping echocardiogram.

To create the heat map and solve the punish method of the model, some methods needed to be tried until the best one was found. The specifics of each method will be explained further, but there are some techniques that are used in both approaches and will be delved into first.

Pixel expansion: As specified in Section 3.6, it is not necessary that a landmark is predicted in the exact coordinate of the "real" landmark, but that does not mean it can be predicted anywhere in the image. So, in order to make the network predict the landmark close to the given one, the acceptable area should be narrowed down. To make that possible, the pixels given by the dataset are "expanded" so that their size is greater than only the one pixel they initially represent.

Border limits: Of course, the heat map cannot be created only by computing the distance between the "real" landmark and the predicted landmark, because it may happen that the network predicts a landmark totally out of the ventricle's border but close to the landmark given, which needs to be harshly punished. In order to reward only the predictions that are inside or really close to the ventricle's border, we need to delimit the border so that the network would not predict any landmark outside of that area.

In order to do this, we used two morphological operations included in OpenCV library: dilation and erosion. The way erosion works is simple, a kernel slides through a given image and the pixel in the original image will be considered 1 only if all the pixels under the kernel are also 1, if not, then the pixel's value will be 0. In an analogous way, dilation slides a kernel though an image, but the pixel inside that image will be considered 0 only if all the pixels under the kernel are also 0, otherwise, its value will be 1.

So, the way these two operations help us is that first, the mask given by the dataset is dilated, making it bigger and the, it is eroded, making it smaller than before. Then, the logical operator XOR is used to compare both images in order to create a final image, which takes the value of 1 only if a specific pixel takes 1 as value only in one image. The same result could be obtained by computing the difference between the two images created by the morphological operations.

Euclidean distance: In this approach, the heat map is calculated only by computing the euclidean distance between the landmarks and every other pixel in the image. The euclidean distance is calculated by the following formula:

$$Euc = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

A map is calculated for every landmark used and is then multiplied by the maps created by the pixel expansion and the border limits. That way, the resulting map will take gradually decreasing values only in pixels that are in the border and close to the original coordinate given by the dataset, which will be the acceptable area to make a prediction. The final result can be seen in Fig 7.

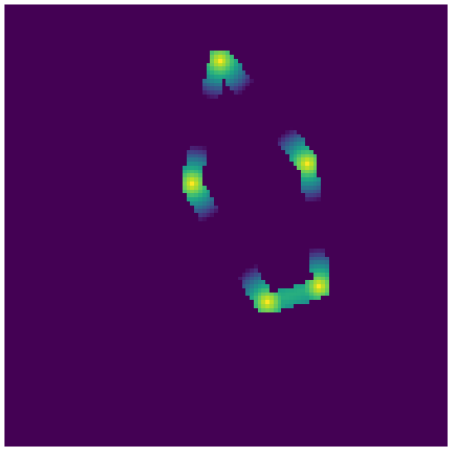

Fig. 7. Heat map using euclidean approach.

Gaussian Distribution: This approach applies a Gaussian filter to the image using the Scikit-Image library's function named "gaussian" (Van der Walt et al., 2014). The Gaussian filter is a kernel that is called in a certain area and makes the central pixel take the highest value and, as the pixels draw away from the center, they take lowest values, using a Gaussian distribution. Once the map is generated, the same process as before is taken: the map is multiplied by the pixel expansion and the border limit maps, enclosing the area in which the model should make a prediction. The heat map can be seen in Fig 8.

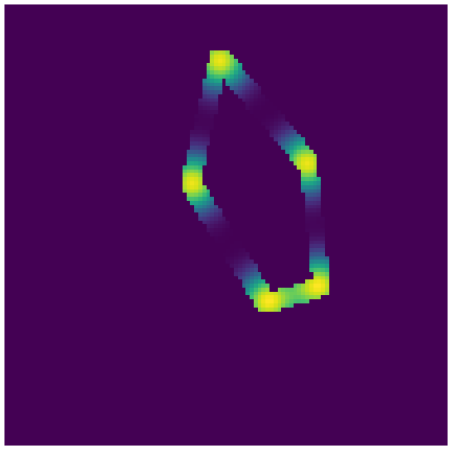

Fig. 8. Heat map using gaussian distribution approach.

Optimal landmarks: As explained in Section 3.6, the number of landmarks is something to think about, as every landmark adds depth to the problem in order to have a (in some times slightly) more accurate mask according to the points received. While the number of landmarks used can be changed as preferred, the most significant landmarks should be obtained in some way in order to get the best mask out of any number of points.

The method chosen to select the most significant landmarks is by calculating the angle between two lines: the first one created by linking one initial landmark and the previous point and the second line created by joining the initial landmark and the next one.

The first step to implement this is ordering the landmarks using the "Centroid method" explained in Section 3.5. Once this is done, every landmark is evaluated by calculating the angle between the lines created by linking said landmark with the previous landmark and the next one. The landmarks with the smallest angle are said to be the most important ones, as an angle close to 180° can be ignored without losing important information.

4 Experimentation and results

In this section, the training methods, results and its interpretation is shown. Every experiment was run in the system described in Section 3.1.

4.1 Experiment 1

In this experiment, the data set was split into 80% training and 20% test. This split results into 16038 images for training and 4010 for testing.

The image segmentation was done using the mask approach, so the input was the original image and the masks created using the methods explained in Section 3.5 and the output, the predicted mask, which was then compared with the original one using the Dice Score. The model was trained over 20 epochs.

4.2 Experiment 2

Since we noted that the model created with the mask approach seems to behave well with few epochs, we decided to train using only 5 epochs and 2000 images split into 80% training and 20% test, resulting in 1600 images for training and 400 for testing.

4.3 Experiment 3

In this experiment, 1,000 images from the data set were split into 80% training and 20% test. This split results into 800 images for training and 200 for testing and the model was trained over 5 epochs (this low amount of epochs was chosen because of the long time this method takes for training) using the landmark approach.

4.4 Experiment 4

In this last experiment, the model was trained with the same setting used in experiment 2 (Section 4.2), but changing the architecture a little. The convolutional layers that link the contracting path and the expansive path were removed. This was done because a simpler model may help solve some possible overfitting noted in previous experiments.

4.5 Results

The results of the first experiment through all the training epochs are shown in Fig. 9. At first, the loss and dice score behave as expected, that is, improving over time, but after a few epochs, the train metrics keep improving, while the validation metrics get stuck in place. That could be a sign of overfitting, but, even if the model does not improve anymore, it ends up with an acceptable Dice Score (0.9946 for the training set and 0.9862 for the validation set).

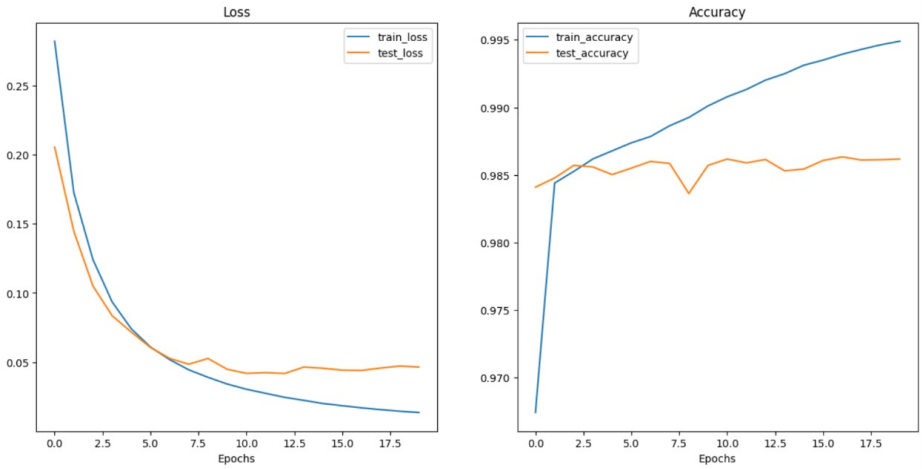

Fig. 9. Experiment 1 results.

In the second experiment, the results of which are shown in Fig. 10, we found that the Dice Score using less epochs and images is not much smaller than the one shown in the first experiment, but the loss function is, in fact, bigger. Another fact to take into account is that the runtime was drastically reduced, while the performance seems to be similar to the one shown by the model created using 10 times more data, but in this experiment we found something interesting that may be affecting the way the model's performance is perceived.

As explained in Section 3.5, the dice score compares how similar two images are, in our specific case, the real mask and the predicted mask, and shows the results using a percentage of this similarity. The problem with this process is

that, in our data set, the pixels that are not part of the left ventricle are more than the pixels that are part of our area of interest. Because of this unbalance, the model can predict a totally black image and still get around 85% Dice Score, making an objectively bad prediction seem like an acceptable one.

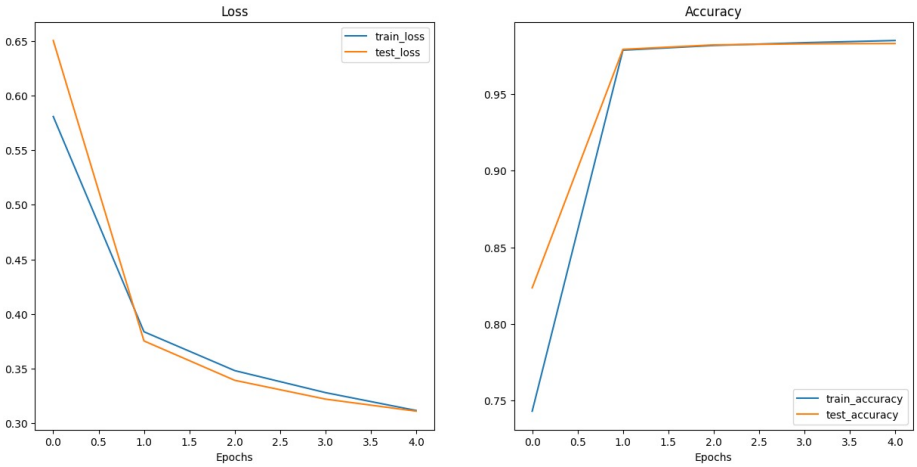

Fig. 10. Experiment 2 results.

The third experiment, which segmented the images using landmarks, showed results (that can be seen in Fig. 11) clearly worse than the mask-based segmentation models in terms of dice score.

Another matter to take in to account is that he loss was really low, but in this case this is no good signal, as it may show that the loss function used to train this model is not appropriate and it may be the reason why the model does not show higher results, as in the backpropagation, the model will not make weight adjustments correctly.

The last experiment's results, which can be seen in Fig. 12, where really similar to the ones obtained in experiment 2, so there is no real advantage in simplifying the architecture in terms of both dice score or the loss function.

Table 1. Experiments results.

	Train Dice Score	Train Loss	Test Dice Score	Test Loss	Time
Experiment 1	0.9946	0.0136	0.9862	0.0465	70 min 57 s
Experiment 2	0.9848	0.3118	0.9828	0.3111	7 min 16 s
Experiment 3	0.0672	0.0611	0.2194	0.0655	21 min 34 s
Experiment 4	0.9909	0.2008	0.9796	0.2148	6 min 49 s

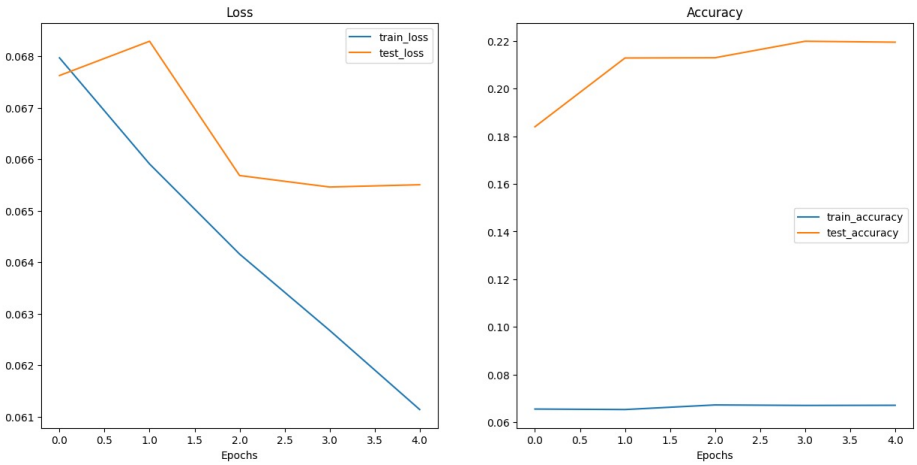

Fig. 11. Experiment 3 results.

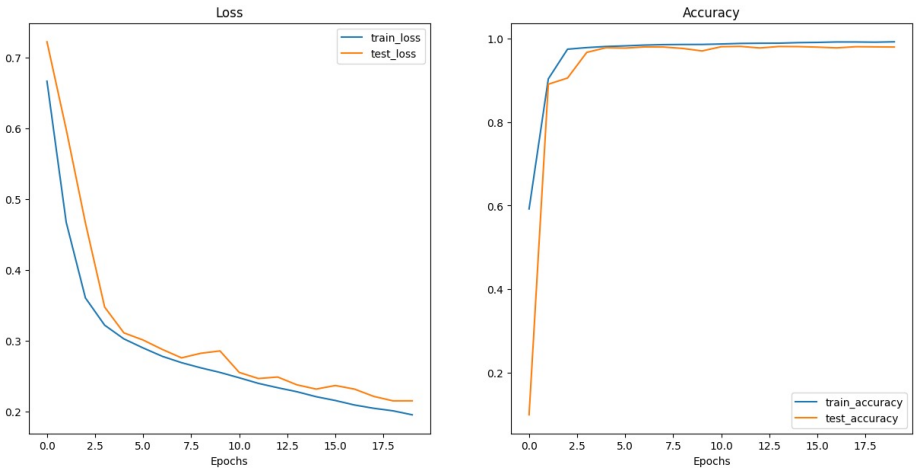

Fig. 12. Experiment 4 results.

5 Conclusions and future work

As seen in Section 4.5, while the model's evaluation metrics seems to be acceptable, this metrics could be misleading, as they tend to show a higher than expected value in cases where the predictions are unreliable. As explained in Section 3.5, image segmentation is a pixel by pixel classification problem and any kind of unbalance may bias the model towards predicting the most recurrent class or, it may be the case that the model labels every pixel as one kind and still get a test metric close to perfection. While this specific case did not happen to the models presented in this paper, it is of interest to solve this problem in future work, as it may lead to a model that extrapolates data better than the current one. The way to solve or, at least, minimize this problem is to implement a loss function that includes class weights in order to slightly bias the algorithm towards predicting more white pixels, as if the class had more presence in the data set.

Another step that could be taken is using a metric similar to dice score, but considering only white pixels in the comparison. This should lead to an easier to understand, more relevant and reliable metric for researchers to see and, while the metric itself would not improve the model as it does not have any influence in the weights learning, once it is created it could be modified to create a new loss function that should be experimented with, the same way the dice score's complement is used to create the dice loss.

One last possible step in order to find the best possible model, is to implement attention layers, which can be used to see why the model takes the decision it takes, as well as marking the image fraction where the model should give more weight to the predictions made. This could be done to have a better understanding of where the models pay attention in order to make decisions and solve the problem of class unbalancing, as the user could point the image's fragment where the left ventricle usually is and give more weight to that predictions.

References

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., ... Farhan, L. (2021). Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1). doi: https://doi.org/10.1186/s40537-021-00444-8

Berman, M., Tupper, C., & Bhardwaj, A. (2022, Sep). *Physiology, left ventricular function - statpearls - ncbi bookshelf.* StatPearls. Retrieved from https://www.ncbi.nlm.nih.gov/books/NBK541098/

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Jadon, S. (2020, October). A survey of loss functions for semantic segmentation. In *2020 ieee conference on computational intelligence in bioinformatics and computational biology (cibcb).* IEEE. Retrieved from http://dx.doi.org/10.1109/CIBCB48159.2020.9277638 doi: https://doi.org/10.1109/cibcb48159.2020.9277638

Long, J., Shelhamer, E., & Darrell, T. (2015, June). Fully convolutional networks for semantic segmentation. In *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr).*

Nasim, M. A. A., Munem, A. A., Islam, M., Palash, M. A. H., Haque, M. M. A., & Shah, F. M. (2023). *Brain tumor segmentation using enhanced u-net model with empirical analysis.*

Nasr-Esfahani, M., Mohrekesh, M., Akbari, M., Soroushmehr, S. M. R., Nasr-Esfahani, E., Karimi, N., ... Najarian, K. (2018). Left ventricle segmentation in cardiac mr images using fully convolutional network. In *2018 40th annual international conference of the ieee engineering in medicine and biology society (embc)* (p. 1275-1278). doi: https://doi.org/10.1109/EMBC.2018.8512536

O'Shea, K., & Nash, R. (2015). *An introduction to convolutional neural networks.*

Ouyang, D. (2020). Video-based ai for beat-to-beat assessment of cardiac function. *Nature*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, & A. F. Frangi (Eds.), *Medical image computing and computer-assisted intervention – miccai 2015* (pp. 234–241). Cham: Springer International Publishing.

Shemona, J. S., & Chellappan, A. K. (2020). Segmentation techniques for early cancer detection in red blood cells with deep learning-based classifier—a comparative approach. *IET Image Processing*, *14*(9), 1726-1732. Retrieved

from https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-ipr.2019.1067 doi: https://doi.org/https://doi.org/10.1049/iet-ipr.2019.1067

Van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., ... Yu, T. (2014). scikit-image: image processing in python. *PeerJ*, *2*, e453.