

Configuring existing software using MRI images

Senior Project II

Spring 2022

Nazarbayev University

Team members: Zhanarys Khorshat, Nursultan Ryspanbetov, Karim Almas, Altynbek
Akaidar

Project advisor: Martin Lukac

Table of contents

Executive summary	3
Introduction	4
Background and Related Work	4
Project approach	5
Project execution	7
Evaluation	14
Conclusion and future work	14
References	15
Appendix.	16

1. Executive summary

The main goal of the project was to configure existing machine learning algorithms to detect knee injury ACL(Anterior Cruciate Ligament) in Magnetic Resonance Imaging(MRI). Moreover, there are some minor goals including: calculation of statistical data, visualization and graphing results. Accomplishing the main goal of the project includes helping NUSOM representatives to get them familiar with the working of algorithms and explain the model's functioning. Therefore, the main user of our product is Nazarbayev University School of Medicine representatives that will use machine learning algorithms. Product of our project is fully functional ML software with graphing representation of model's parameters such as AUC, train loss after model's training in TensorFlow. Additionally, the product includes a widget that gives the user graphical view of an MRI image slice by slice in one of the axes (sagittal,coronal,axial) and a widget that shows results of the model on a validation set with visual view on MRI images. All coding was done using Python language and using open source libraries such as TensorFlow and Plotly for visualization.

2. Introduction

In the 21st century, modern technologies in medicine are not only the latest medical equipment, but also industry-specific software that automates all work processes. However, even if technologies such as Computed Tomography(CT) and Magnetic Resonance Imaging(MRI) allow doctors to evaluate different human parts, as well as to determine the causes of abnormalities in various diseases, there are still certain drawbacks. With manual data processing, the doctor needs a lot of time to assess the picture. Furthermore, there is, although not a significant percentage, at which the diagnosis may be incorrect. To illustrate, in a study conducted in 2014, it is demonstrated that a large number of errors were detected in conventional radiographs: CT 30.5% and MRI 11.4%(Kim and Mansfield, 2014). Nonetheless, today, considering more computing power and improved quality of medical data, the use of artificial intelligence methods is generating growing interest in the visualization of human organs. Moreover, machine learning could automatize detection of diseases in magnetic resonance images by using pre-trained models that will save time of processing data for doctors and could solve the problem of lack of specialists. However, ML models may not have high accuracy in detecting injuries as radiologists but over time the model can reach higher accuracy as it will gain more and more data that contain different cases.

Our solution is to configure existing models that use convolutional neural networks implemented in PyTorch to help NUSOM representatives to run the model on their own and explain to them the general structure and workflow of the project. In general, firstly, we configure existing softwares to state “ready-to-go”. Then, present software for NUSOM representatives and get feedback. Finally, we prepare proper visualization to use them in their research works including sensitivity and specificity scores.

In this report, you will get acquainted with the software that is trained on a given dataset, and then predicts diseases automatically. In the report, firstly, background of the project will be discussed as related works will be examined. Then, the report includes our project approach including all tools and features of the project. Also, the report provides work done over two last semesters with problems encountered during the project execution. Report finishes with conclusion including evaluation of the project and future project plans.

3. Background and Related Work

Nowadays, it is impossible to imagine medicine without radiation diagnostics. In the first decades of its development, only projection methods were used - radiography, angiography, planar scintigraphy. Technological progress and the advent of computers led to the development of tomographic methods, which today occupy a leading place in radiation

diagnostics. First of all, this applies to computed tomography (CT) and magnetic resonance imaging (MRI). Magnetic resonance imaging (MRI) is a medical technique, which is used for generating pictures of human body parts, anatomy and physiological processes. MRI technology is used to detect tissues that have pathological manifestations, such as tumors. Particularly, with the use of MRI technique, we can diagnose the Anterior Cruciate Ligament (ACL) injury. Anterior Cruciate Ligament (ACL) is one the primary ligaments that stabilizes the knee joint, thereby controlling back and forth movements of the knee. Tears of the ACL occur during rotational (circular) movements of the knee with a fixed foot. With a fresh ligament injury, an obligatory radiography is required to identify concomitant injuries of the knee joint. However, damage to the soft tissue structures is not visible on radiographs. Therefore, the gold standard for non-invasive diagnosis of ACL disease is magnetic resonance imaging.

We analyzed several projects that use ML to detect the anterior cruciate ligament (ACL) disease. After careful study, models with best accuracy were picked:

- <https://github.com/neergaard/MRNet>
- <https://github.com/yashbhalgat/MRNet-Competition>
- <https://github.com/ahmedbesbes/mrnet>

Moreover, we completed literature review to familiarize with other done researches and projects in this field. Reviewed literatures are presented in references. Because the efficiency of the model can be identified only in comparison with others. Study of Kim Y. showed that they had an accuracy in detecting radiological diagnosis was 70%. In the study of Kuze K. N. 11 AI studies were conducted from which 5 investigated ACL tears, 5 investigated meniscal tears, and 1 investigated both. The AUC of AI models for detecting ACL tears ranged from 0.895-0.980, and the prediction accuracy ranged from 86.7-100%. Finally, Namiri N. in his study ACL injury classifications using the 3D convolutional neural networks (CNN) and 2D CNN showed accuracies 89% and 92%, respectively. Both CNNs had a weighted Cohen k of 0.83 and had similar classification of intact ACLs (2D CNN with 93% sensitivity and 90% specificity vs 3D CNN with 89% sensitivity and 88% specificity). Both CNNs performed similarly in classifying full tears. The 2D CNN classified all reconstructed ACLs correctly.

4. Project approach

Projects consist of two main components which are dataset and machine learning model. There are two main datasets that we used to configure the project that come from:

- Stanford University Medical Center
- Clinical Hospital Center Rijeka

Data is represented in two ways: **.pck** file and **.npy** file. Data from Clinical Hospital is a collection of **.pck** files consisting of all information about all 3 planes(*axial,sagittal* and *coronal*) in one single file while data from Stanford is numpy files representing only one single plane. File contains a numpy array with following dimensions (number of slices,3,256,256) as each file has a different number of slices from 20 up to 40. Maximum pooling technique is applied on slices to have a one dimensional array on max-pooling layers of CNN. This is what files consist of:

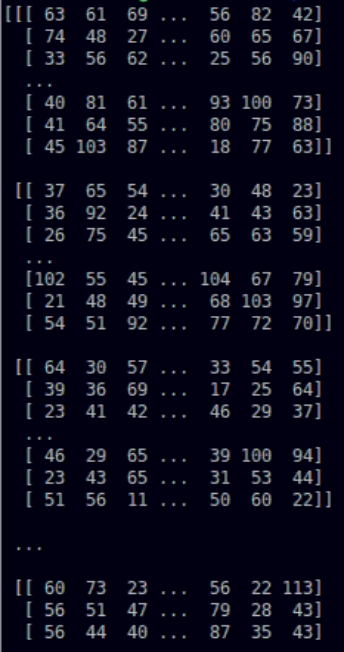


Figure 1. Composition of file.

MRInet models mostly consist of 4 main files: *loader*, *evaluation*, *train* and *model* itself. **Loader** is responsible for loading all data into the model. Data is divided into 3 sets including training, validation and testing. So, the **train** uses a training set to train the model while **evaluation** uses other two sets to examine the model's accuracy and loss.

During the loading of data, the **loader** modifies data by use of data augmentation. It helps to improve the model's performance and prevents it from overfitting by running it on various examples of data. During augmentation, data is randomly rotated and flipped horizontally.

During the training of data, weights of the model are changed over a loop. Training uses an Adam optimizer from a PyTorch and binary cross entropy loss to adjust weights by minimizing the loss. Weights are adjusted in the opposite direction of the gradient that is calculated using backpropagation.

Model is built using pre-trained Alexnet architecture from PyTorch consisting of 8 pooling and convolutional layers ending with fully connected layers. Model architecture is defined in a forward method which means moving from input to output layers.

All project was done in a “conda” environment and used also different supportive Python packages that are listed below:

```
- blas=1.0=mkl
- ca-certificates=2018.03.07=0
- certifi=2018.8.24=py36_1
- cffi=1.11.5=py36he75722e_1
- freetype=2.9.1=h8a8886c_1
- intel-openmp=2018.0.3=0
- jpeg=9b=h024ee3a_2
- libedit=3.1.20170329=h6b74fdf_2
- libffi=3.2.1=hd88cf55_4
- libgcc-ng=8.2.0=hdf63c60_1
- libgfortran-ng=7.3.0=hdf63c60_0
- libpng=1.6.34=h9fc6fc_0
- libstdcxx-ng=8.2.0=hdf63c60_1
- libtiff=4.0.9=he85c1e1_2
- mkl=2018.0.3=1
- mkl_fft=1.0.4=py36h4414c95_1
- mkl_random=1.0.1=py36h4414c95_1
- ncurses=6.1=hf484d3e_0
- ninja=1.8.2=py36h6bb024c_1
- numpy=1.15.1=py36h1d66e8a_0
- numpy-base=1.15.1=py36h81de0dd_0
- olefile=0.45.1=py36_0
- openssl=1.0.2p=h14c3975_0
- pillow=5.2.0=py36heded4f4_0
- pip=10.0.1=py36_0
- pycparser=2.18=py36_1
- python=3.6.6=hc3d631a_0
- readline=7.0=h7b6447c_5
- scikit-learn=0.19.1=py36hedc7406_0
- scipy=1.1.0=py36hfa4b5c9_1
- setuptools=40.2.0=py36_0
- six=1.11.0=py36_1
- sqlite=3.24.0=h84994c4_0
- tk=8.6.8=hbc83047_0
- wheel=0.31.1=py36_0
- xz=5.2.4=h14c3975_4
- zlib=1.2.11=ha838bed_2
- pytorch=0.4.1=py36_py35_py27__9.0.176_7.1.2_2
- torchvision=0.2.1=py36_1
```

Figure 2. Used python packages

To visualize the model, we created python files that use plotly and matplotlib libraries to make graphical representations of each slice in one of the presented axes. To create visualizations of file slice by slice, plotly’s Figure method was used. Also, we used TensorFlow to visualize the loss of a model during training and validation, and it’s AUC in training and validation. Moreover, Class Activation Mapping was used to create a web-based widget on a validation set.

5. Project execution

First of all, we started with studying the Stanford Machine Learning course to better understand how models work(<https://cs231n.github.io/>) as we had a lack of knowledge in this field.

During the senior project to manage our work process and divide roles within a group we used the kanban method in Trello Application. Also, we mostly used the Rapid Application Development process. The main reason is that NUSOM representatives told us what they wanted and then we fulfilled their requirements.

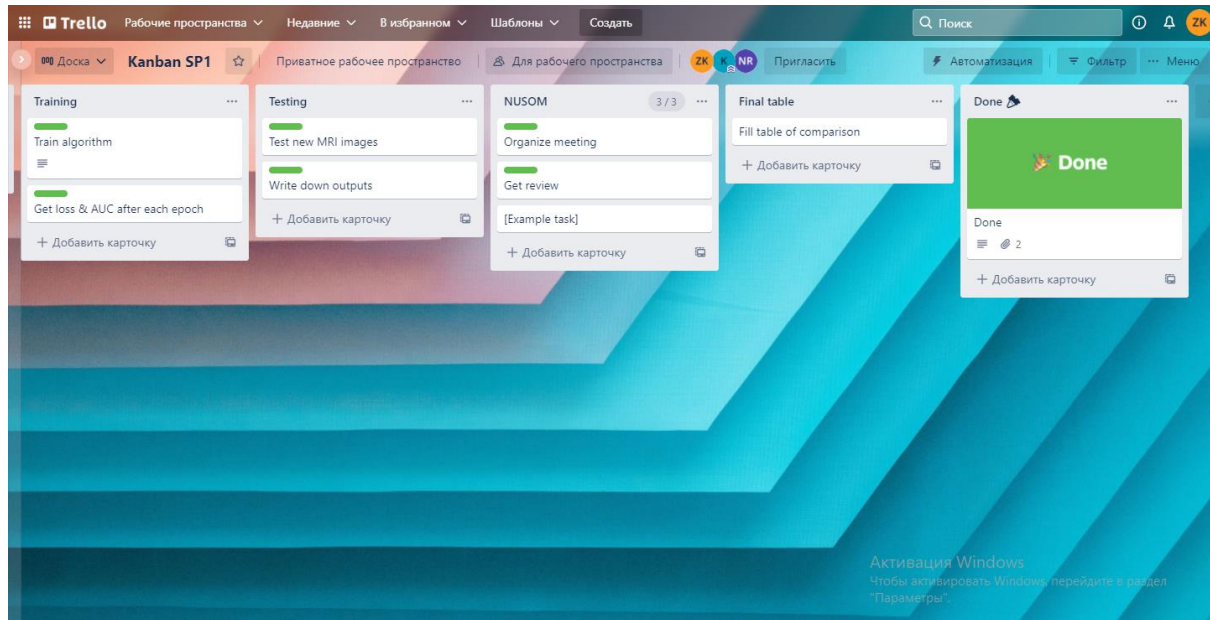


Figure 3. Trello application

We almost achieved all planned goals for both semesters. We successfully configured all three models to “ready-to-go” state:

- Neergard’s model
- Yashbalgat’s model
- Besbes’s model

In the beginning, to run any of the models, users should make environment preparation including setup of conda environment and installing all necessary packages. To make the process of environment setup **environment.yml** was created with all needed packages with appropriate versions. So, users can create a conda environment using the above file and activate it. During project execution, some changes were applied into all three models as there was a problem with successful running.

Results of running models are presented below:

```

Terminal - snake@snake-C4-411:~/Desktop/mrnet/dash
File Edit View Terminal Tabs Help
[Epoch: 13 / 50 [Single batch number : 1100 / 1130 ]] | avg train loss 1.0367 | train auc : 0.8109 | lr : 1e-05
[Epoch: 13 / 50 [Single batch number : 20 / 120 ]] | avg val loss 0.7903 | val auc : 0.8654 | lr : 1e-05
[Epoch: 13 / 50 [Single batch number : 40 / 120 ]] | avg val loss 1.0034 | val auc : 0.8421 | lr : 1e-05
[Epoch: 13 / 50 [Single batch number : 60 / 120 ]] | avg val loss 0.8563 | val auc : 0.8275 | lr : 1e-05
[Epoch: 13 / 50 [Single batch number : 80 / 120 ]] | avg val loss 0.8336 | val auc : 0.8556 | lr : 1e-05
[Epoch: 13 / 50 [Single batch number : 100 / 120 ]] | avg val loss 0.8323 | val auc : 0.866 | lr : 1e-05
train loss : 1.0339 | train auc 0.8095 | val loss 0.8096 | val auc 0.8561 | elapsed time 279.88840134620667 s
-----
[Epoch: 14 / 50 [Single batch number : 100 / 1130 ]] | avg train loss 1.1759 | train auc : 0.7808 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 200 / 1130 ]] | avg train loss 1.1137 | train auc : 0.7605 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 300 / 1130 ]] | avg train loss 1.094 | train auc : 0.7783 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 400 / 1130 ]] | avg train loss 1.1535 | train auc : 0.7648 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 500 / 1130 ]] | avg train loss 1.1062 | train auc : 0.7848 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 600 / 1130 ]] | avg train loss 1.0958 | train auc : 0.797 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 700 / 1130 ]] | avg train loss 1.0816 | train auc : 0.8068 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 800 / 1130 ]] | avg train loss 1.0702 | train auc : 0.7997 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 900 / 1130 ]] | avg train loss 1.0712 | train auc : 0.803 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 1000 / 1130 ]] | avg train loss 1.0546 | train auc : 0.8024 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 1100 / 1130 ]] | avg train loss 1.0406 | train auc : 0.8053 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 20 / 120 ]] | avg val loss 0.9531 | val auc : 0.9727 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 40 / 120 ]] | avg val loss 1.1223 | val auc : 0.9476 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 60 / 120 ]] | avg val loss 1.137 | val auc : 0.8987 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 80 / 120 ]] | avg val loss 1.0594 | val auc : 0.895 | lr : 1e-05
[Epoch: 14 / 50 [Single batch number : 100 / 120 ]] | avg val loss 1.1187 | val auc : 0.8536 | lr : 1e-05
Epoch: 14: reducing learning rate of group 0 to 3.0000e-06.
train loss : 1.0403 | train auc 0.8046 | val loss 1.1255 | val auc 0.8496 | elapsed time 278.05244731903076 s
-----
[Epoch: 15 / 50 [Single batch number : 100 / 1130 ]] | avg train loss 1.0366 | train auc : 0.8567 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 200 / 1130 ]] | avg train loss 0.9996 | train auc : 0.8355 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 300 / 1130 ]] | avg train loss 0.9352 | train auc : 0.8529 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 400 / 1130 ]] | avg train loss 0.9602 | train auc : 0.8562 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 500 / 1130 ]] | avg train loss 0.9592 | train auc : 0.8594 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 600 / 1130 ]] | avg train loss 0.9466 | train auc : 0.862 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 700 / 1130 ]] | avg train loss 0.9785 | train auc : 0.8583 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 800 / 1130 ]] | avg train loss 0.936 | train auc : 0.8675 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 900 / 1130 ]] | avg train loss 0.9399 | train auc : 0.8592 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 1000 / 1130 ]] | avg train loss 0.9475 | train auc : 0.8551 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 1100 / 1130 ]] | avg train loss 0.9434 | train auc : 0.855 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 20 / 120 ]] | avg val loss 0.6966 | val auc : 0.9352 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 40 / 120 ]] | avg val loss 0.7884 | val auc : 0.8952 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 60 / 120 ]] | avg val loss 0.8096 | val auc : 0.9118 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 80 / 120 ]] | avg val loss 0.7477 | val auc : 0.9054 | lr : 3e-06
[Epoch: 15 / 50 [Single batch number : 100 / 120 ]] | avg val loss 0.907 | val auc : 0.8518 | lr : 3e-06
train loss : 0.9432 | train auc 0.8558 | val loss 0.8568 | val auc 0.8611 | elapsed time 276.8653326034546 s
-----
Early stopping after 5 iterations without the decrease of the val loss
training took 4710.061040878296 s
(base) snake@snake-C4-411:~/Desktop/mrnet$ cd da

```

Figure 4. Running AhmedBesbes's model

```

Terminal - snake@snake-C4-411:~/MRNet
File Edit View Terminal Tabs Help
starting epoch 45. time passed: 0:20:31.997779
train loss: 0.0121
train AUC: 0.9996
valid loss: 0.0251
valid AUC: 0.9248
starting epoch 46. time passed: 0:20:59.758666
train loss: 0.0121
train AUC: 0.9996
valid loss: 0.0250
valid AUC: 0.9248
starting epoch 47. time passed: 0:21:27.828079
train loss: 0.0120
train AUC: 0.9996
valid loss: 0.0250
valid AUC: 0.9248
starting epoch 48. time passed: 0:21:55.835635
train loss: 0.0120
train AUC: 0.9996
valid loss: 0.0251
valid AUC: 0.9248
starting epoch 49. time passed: 0:22:23.468029
train loss: 0.0120
train AUC: 0.9995
valid loss: 0.0253
valid AUC: 0.9248
starting epoch 50. time passed: 0:22:51.514172
train loss: 0.0120
train AUC: 0.9996
valid loss: 0.0250
valid AUC: 0.9248
(mrnet) snake@snake-C4-411:~/MRNet$

```

Figure 5. Running Neergard's model

```

Terminal - snake@snake-C4-411: ~/MRNet-Competition/src
File Edit View Terminal Tabs Help
train loss: 0.0673
train AUC: 0.9888
100% | 120/120 [00:14<00:00, 8.18it/s]
valid loss: 0.1305
valid AUC: 0.9457
starting epoch 45, time passed: 1:55:55.197047
100% | 1130/1130 [02:23<00:00, 7.87it/s]
train loss: 0.0673
train AUC: 0.9888
100% | 120/120 [00:14<00:00, 8.14it/s]
valid loss: 0.1305
valid AUC: 0.9457
starting epoch 46, time passed: 1:58:33.612591
100% | 1130/1130 [02:22<00:00, 7.93it/s]
train loss: 0.0672
train AUC: 0.9889
100% | 120/120 [00:14<00:00, 8.20it/s]
valid loss: 0.1305
valid AUC: 0.9461
starting epoch 47, time passed: 2:01:10.729040
100% | 1130/1130 [02:24<00:00, 7.83it/s]
train loss: 0.0672
train AUC: 0.9889
100% | 120/120 [00:14<00:00, 8.27it/s]
valid loss: 0.1304
valid AUC: 0.9461
starting epoch 48, time passed: 2:03:49.546942
100% | 1130/1130 [02:23<00:00, 7.88it/s]
train loss: 0.0672
train AUC: 0.9889
100% | 120/120 [00:14<00:00, 8.13it/s]
valid loss: 0.1304
valid AUC: 0.9461
starting epoch 49, time passed: 2:06:27.649960
100% | 1130/1130 [02:23<00:00, 7.87it/s]
train loss: 0.0672
train AUC: 0.9889
100% | 120/120 [00:14<00:00, 8.22it/s]
valid loss: 0.1305
valid AUC: 0.9461
starting epoch 50, time passed: 2:09:05.829040
100% | 1130/1130 [02:23<00:00, 7.88it/s]
train loss: 0.0672
train AUC: 0.9889
100% | 120/120 [00:14<00:00, 8.05it/s]
valid loss: 0.1305
valid AUC: 0.9461
(snake) snake@snake-C4-411:~/MRNet-Competition/src$

```

Figure 6. Running Yashbalgat's model

Based on the results of the three models, a comparison table was created that includes training AUC, validation AUC, loss on training and loss on validation. Besbes's model run on single axis, so we put results of the model on each axis.

Model	Training AUC	Validation AUC	Training loss	Validation loss	Training time, minutes
Neergard	0.9996	0.9248	0.0120	0.0250	22.8
Besbes Sagittal	0.8799	0.8979	0.1055	0.1157	106.2
Besbes Coronal	0.8994	0.8496	0.2997	0.057	88.5
Besbes Axial	0.886	0.8875	0.03277	0.3218	53.1
Yashbalgat	0.980	0.950	0.0672	0.1305	130.8

Table 1. Comparison table of model's results

Also, figures of training loss, validation loss, training AUC, validation AUC versus iterations were made. You can see them in a Appendix

We made a visualization of numpy files to be able to see each axis of data(coronal,axial,sagittal) slice by slice. User specifies the axis and filename then web-based visualization is shown with some instruments. Interface allows users to view each in play and go mode. Also, users can zoom in and out by stopping at a certain slice.

Slices in volumetric data

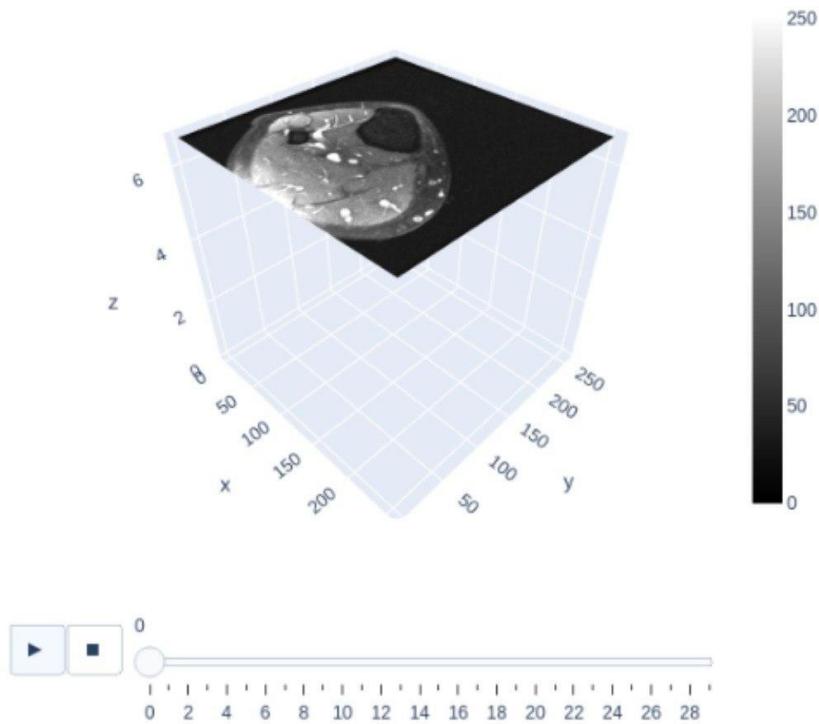


Figure 7. Web-based visualization of one axis

Also, a program to show and save every slice of some file in a given axis was made by request of NUSOM members. By executing a program, the user specifies file, axis and slice amount and receives all slices in a png file format in a certain directory.

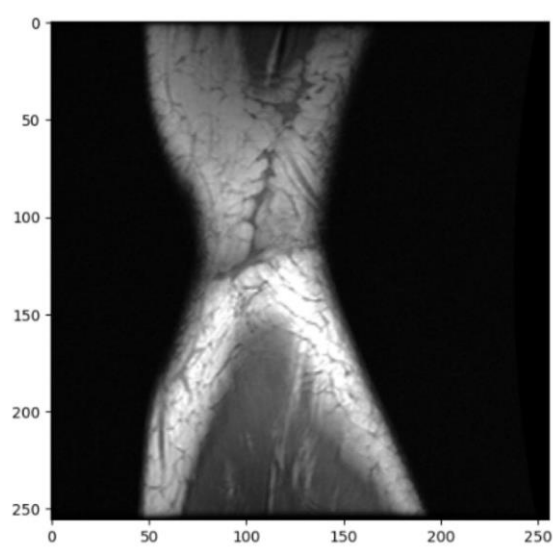


Figure 8. Representation of png file.

Moreover, one more web-based visualization was created to visualize the results of the model on a validation set. Users can see images that were correctly diagnosed with probability values and images which were diagnosed incorrectly. By this NUSOM

representatives can see calculated probabilities for each file. Also, visual representation of MRI image slice by slice is also added to this interface to investigate cases of correctly and incorrectly diagnoses. For example, case 1200 was correctly diagnosed by model with probability of 0.8194.

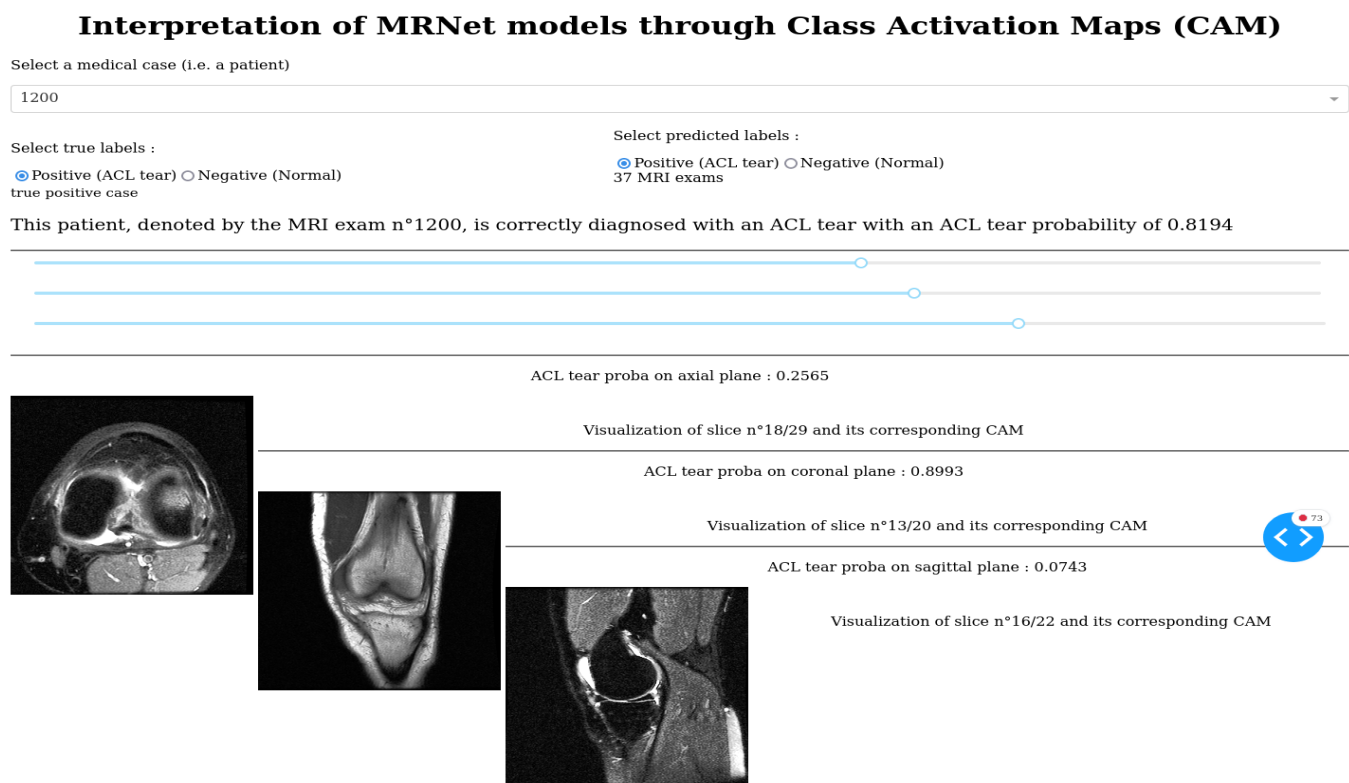
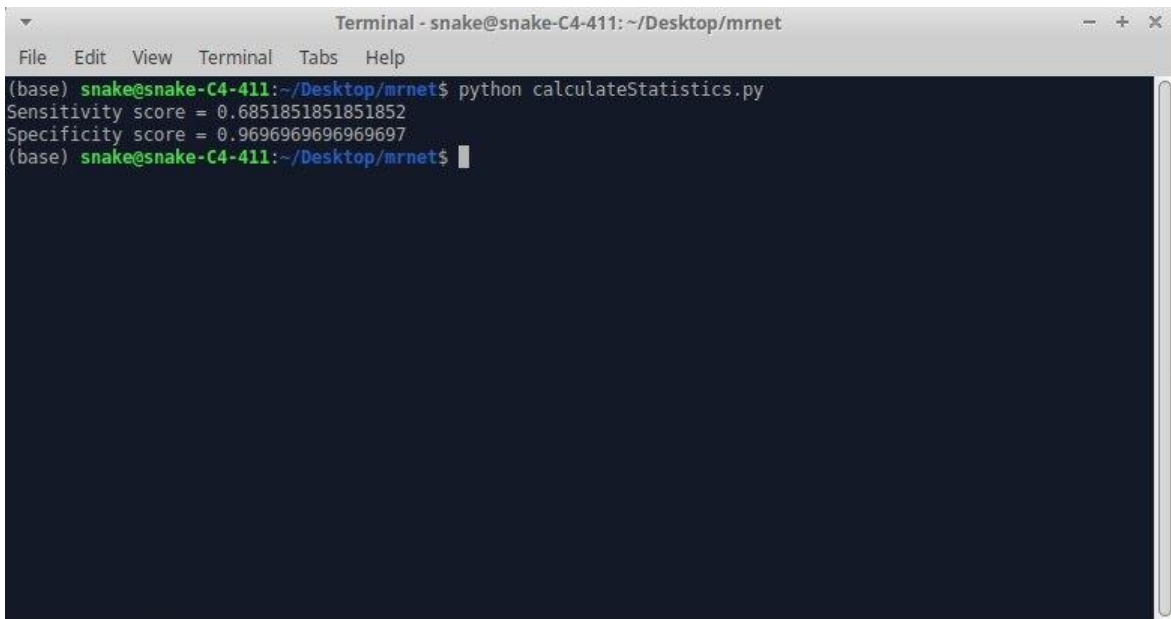


Figure 9. Interpretation of MRNet models through CAM

Next, users can get statistical data including specificity and signficity scores of the model after running the model on a validation set. User just needs to run the program and it calculates scores using predicted values of the model and shows results in a console.

A terminal window titled "Terminal - snake@snake-C4-411: ~/Desktop/mrnet" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the command `python calculateStatistics.py` being executed. The output displays two lines: "Sensitivity score = 0.6851851851851852" and "Specificity score = 0.9696969696969697". The prompt `(base) snake@snake-C4-411:~/Desktop/mrnet$` is visible at the bottom.

```
Terminal - snake@snake-C4-411: ~/Desktop/mrnet
File Edit View Terminal Tabs Help
(base) snake@snake-C4-411:~/Desktop/mrnet$ python calculateStatistics.py
Sensitivity score = 0.6851851851851852
Specificity score = 0.9696969696969697
(base) snake@snake-C4-411:~/Desktop/mrnet$
```

Figure 10. Sensitivity and specificity scores for besbes's model

As running models or running programs to get a visual interface is not something easy for a person without coding experience, we created a README file in markdown format. File includes step-by-step instructions to execute any program. By simply following instructions, a person can get all the results that were listed above.

Problems encountered

During project execution there were many changes entered. Main reason is that we changed our aims by request of NUSOM members.

Main problem that we addressed was lack of knowledge in machine learning. We are still managing that issue by learning short ML courses and searching for exact coding problems in sites like stackoverflow and so on. Moreover, at the beginning of the project, we encountered the problem of the absence of appropriate PC devices as software needs a high computational power (GPU) and proper operating system (Ubuntu) to get all needed packages. However, it was solved fast as we got access to a PC in the laboratory that meets all system requirements.

All project components are available by following link:

<https://github.com/Jonerbay/Senior-Project>

All data are available by following link:

<https://drive.google.com/drive/folders/1mpSe2ONoq6XUTHMdUS9ci7V3XNKJjfYu>

6. Evaluation

As our main and only user is NUSOM members. We presented all the completed work with instructions to them to get evaluation from them. We asked them some questions and gave a brief commentary on the question.

1. Was it easy to follow instructions given in README.md?
2. Did you encounter any problems with running models and widgets?
3. Do you have any suggestions to improve the user orientation of the project?
4. Was all goals of the project accomplished?

Answers to the questions in unchanged form:

1. "Instructions are well-made and relatable which makes it easy to follow. However, I think you can add some step guides or paste links about installation of conda."
2. "Overall, I did not encounter any huge problem but it was hard for me to run display.py and printSlices.py files as I was slightly confused with the project paths. I think you can add to instructions"
3. "As I said, you can add some extra things to the instructions. It was difficult and time consuming to download Stanford data from the drive. It will be good if you create a program to download Stanford data as quickly as Istajduh's data using download.sh file."
4. "Most of the goals were accomplished except 3d Visualization".

7. Conclusion and future work

Most of the tasks were accomplished during the whole project. However, we did not finish 3d visualization of the MRI image by this deadline.

There are main tasks that were done:

- Explained how the algorithm works and the model's functioning to the NUSOM representatives.
- A Machine Learning software with full functionality and graphical representation with parameters such as AUC
- The widget that shows graphical representation of MRI image slice by slice
- The widget that shows results of the model on a validation set with visual view on MRI images

Possible work in the future and recommendations:

- Finish the 3D visualization
- Collect feedback from experienced researcher to follow their suggestions

8. References

- Kim, Y. W., & Mansfield, L. T. (2014). Fool me twice: delayed diagnoses in radiology with emphasis on perpetuated errors. *American journal of roentgenology*, 202(3), 465-470.
- Kunze, K. N., Rossi, D. M., White, G. M., Karhade, A. V., Deng, J., Williams, B. T., & Chahla, J. (2021). Diagnostic performance of artificial intelligence for detection of anterior cruciate ligament and Meniscus Tears: A systematic review. *Arthroscopy: The Journal of Arthroscopic & Related Surgery*, 37(2), 771–781. <https://doi.org/10.1016/j.arthro.2020.09.012>
- Namiri, N. K., Flament, I., Astuto, B., Shah, R., Tibrewala, R., Caliva, F., Link, T. M., Pedoia, V., & Majumdar, S. (2020). Deep learning for hierarchical severity staging of anterior cruciate ligament injuries from MRI. *Radiology: Artificial Intelligence*, 2(4). <https://doi.org/10.1148/ryai.2020190207>

Appendix.

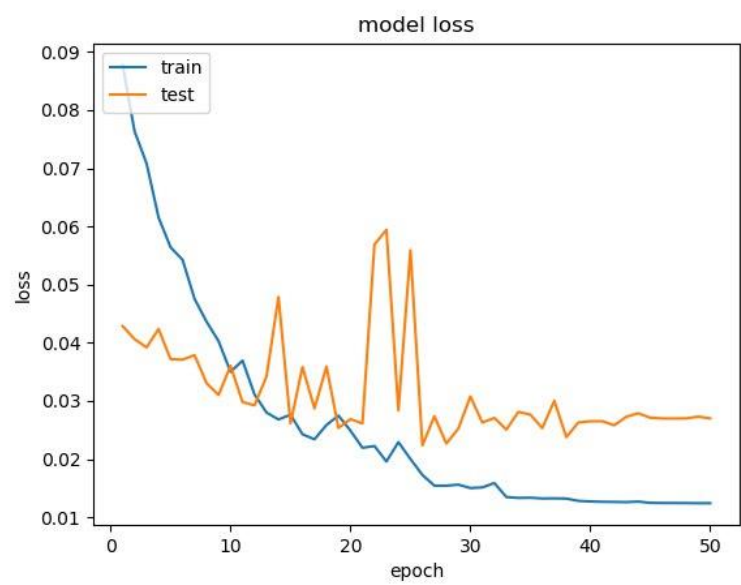


Figure A. Loss Models

	Name	Smoothed	Value	Step	Time	Relative
●	acl/axial/20220325-170619	0.8819	0.886	12.43k	Fri Mar 25, 18:05:35	59m 3s
●	acl/coronal/20220328-144949	0.8998	0.8994	22.6k	Mon Mar 28, 16:23:02	1h 33m 1s
●	acl/sagittal/20220327-172733	0.8837	0.8799	20.34k	Sun Mar 27, 19:18:30	1h 50m 47s

Figure B. Ahmed Besbes AUC train

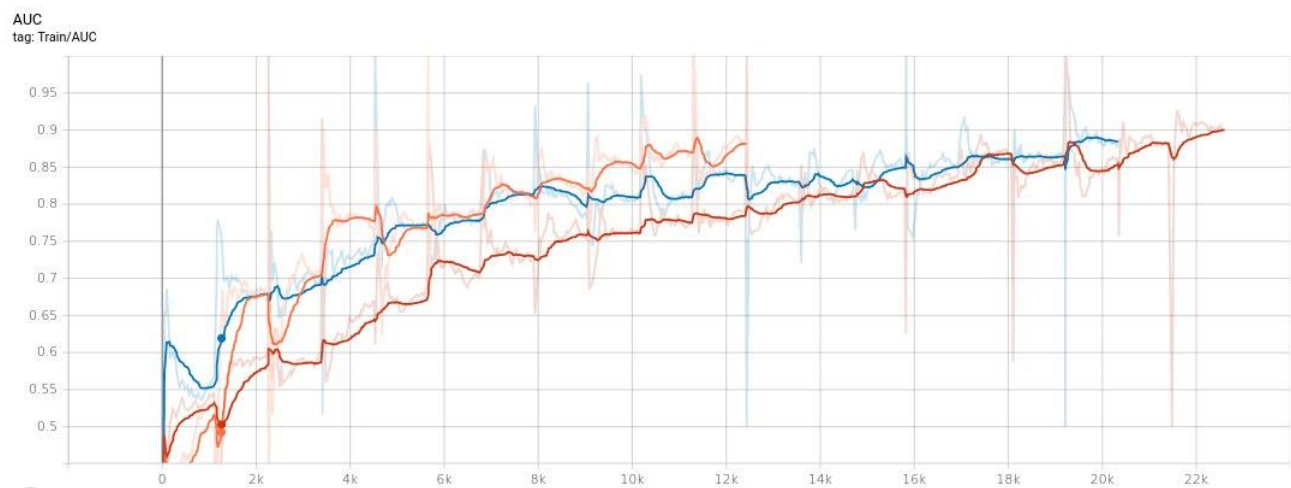


Figure C. Train vs. AUC graph

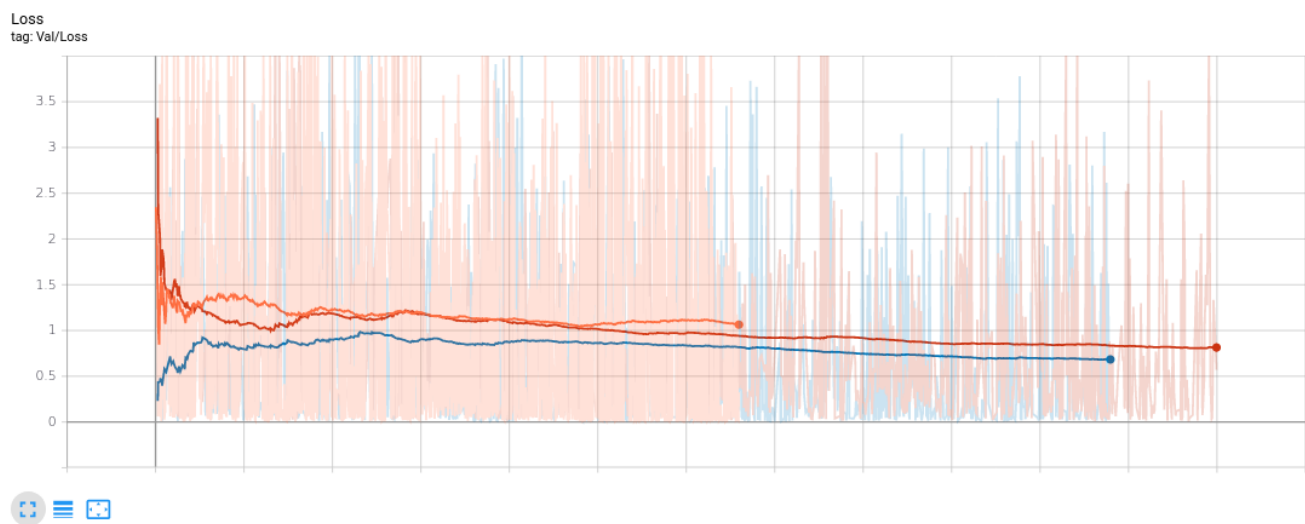
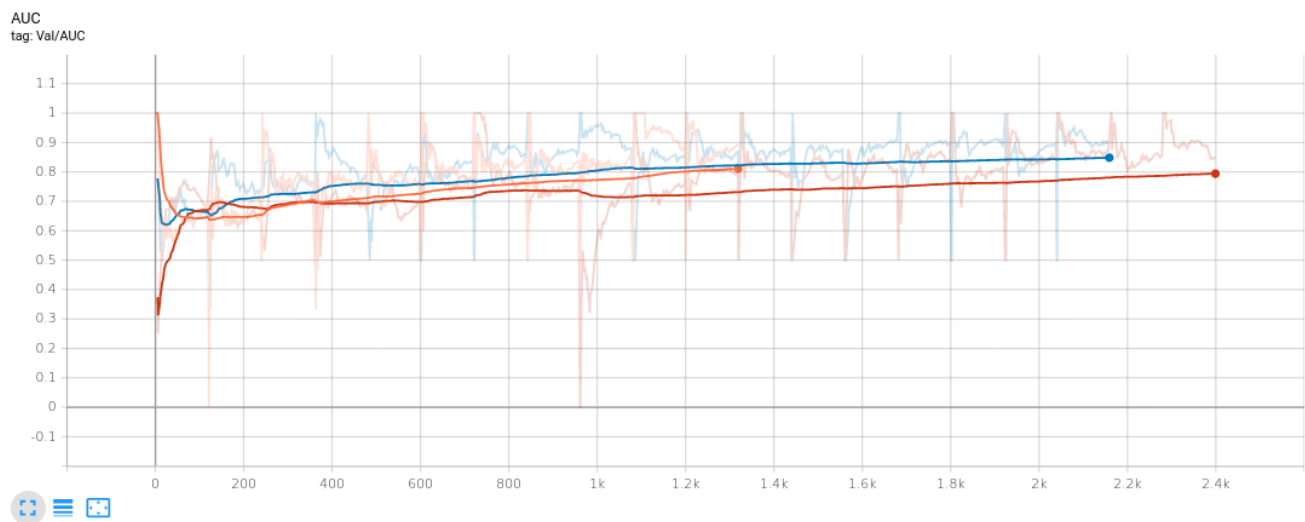


Figure D. Train vs. Loss graph



Name	Smoothed	Value	Step	Time	Relative
axial/20220325-170619	0.5753	0.03277	12.43k	Fri Mar 25, 18:05:35	59m 3s
coronal/20220328-144949	0.6862	0.2997	22.6k	Mon Mar 28, 16:23:02	1h 33m 1s
sagittal/20220327-172733	0.966	1.055	20.34k	Sun Mar 27, 19:18:30	1h 50m 47s

Figure F. Ahmed Besbes Loss train

Name	Smoothed	Value	Step	Time	Relative
axial/20220325-170619	0.553	0.3218	1.319k	Fri Mar 25, 18:05:40	53m 57s
coronal/20220328-144949	0.836	0.57	2.399k	Mon Mar 28, 16:23:05	1h 28m 38s
sagittal/20220327-172733	0.6025	0.1157	2.159k	Sun Mar 27, 19:18:40	1h 46m 17s

Figure G. Ahmed Besbes Loss val

Name	Smoothed	Value	Step	Time	Relative
axial/20220325-170619	0.8761	0.8875	1.319k	Fri Mar 25, 18:05:40	53m 57s
coronal/20220328-144949	0.8803	0.8496	2.399k	Mon Mar 28, 16:23:05	1h 28m 38s
sagittal/20220327-172733	0.8994	0.8979	2.159k	Sun Mar 27, 19:18:40	1h 46m 17s

Figure H. Ahmed Besbes AUC val

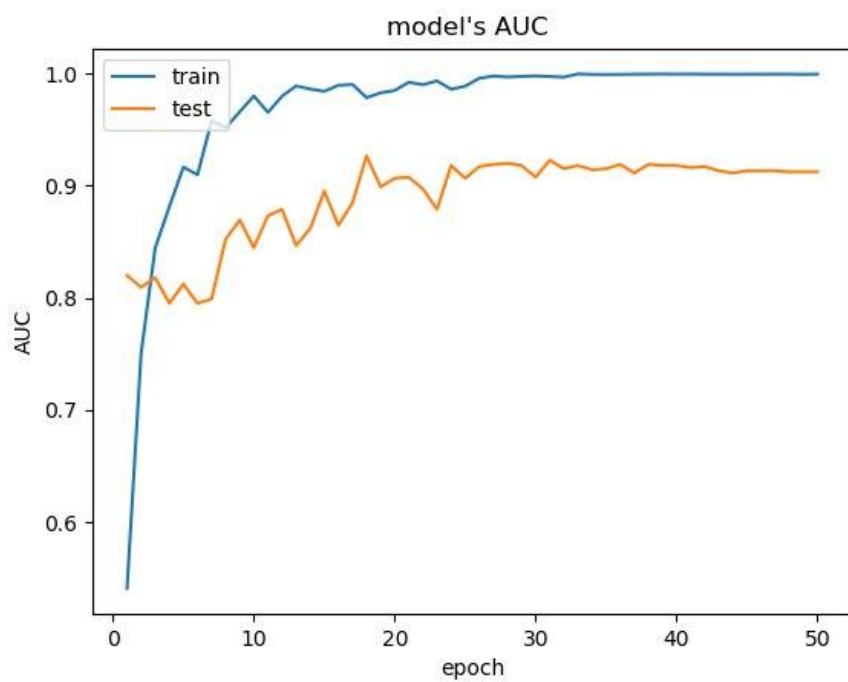


Figure I. epoch vs. AUC graph.

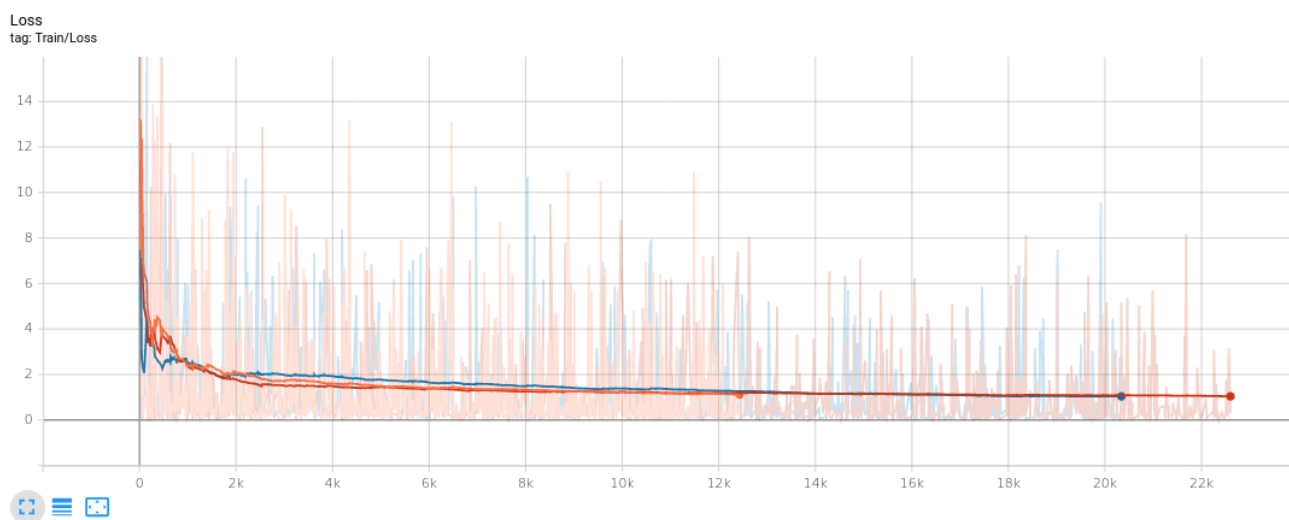


Figure J. Loss vs. Train graph.