Abstract:

Introduction:

Problem statement:

The problem required to be solved is at its root an image recognition problem (more precisely image classification). Given as input three sets (slices) of image series for a patients MRI knee examination, the required is to train a neural network model that diagnoses the examination and gives three outputs. The first is the probability of the patient having abnormalities in his knee. The second and third further classify these abnormalities showing the probability that the patient has ACL tears or meniscal tears respectively. This is done by building the MRnet model and training it by the help of the MRnet data set using deep learning techniques.

Objective:

Our main objective is implementing the deep learning neural network *MRnet*. This is done by trying the paper's approach using different building blocks than those used in the paper. Mainly by using feature extractors based on architectures other than AlexNet that was used in the paper's original implementation and comparing the obtained results. These included other sub objectives as building the model architectures from scratch and using transfer-learning techniques.

Dealing with the data set:

The data used was the same data set used by the paper in training the MRnet model except for the following changes:

- The data provided is in the form of numpy arrays with .npy extension, where each array is of shape sx256x256. This is in contrast to the images input referred to in the paper of shape sx3x256x256.
- In order to obtain the input data in image like format, each input array was stacked three times to obtain a grey-scaled image representation of the MRI. Notice that here the image has same pixel values in red, green and

blue channels as the same array is stacked three times, giving a grey-scaled image.

- The input dimensions after stacking is sx256x256x3 instead of sx3x256x256, which is not a great difference it is just using a different channels convention (channels-last) than that stated in the paper (channels-first). This is only for convenience in implementation of the model.
- When training the extractors, the first image only from each series was used. Further elaboration and justification is given in a following section clarifying our approach.
- The test data used in the paper is not obtainable so we used the validation data in testing and the training data was split at a ratio of 90% to 10% to training data and validation data.
- The training input was normalized to contain values from 0 to 1 instead of having the full pixel value ranges (0 to 255).
- Data augmentation was used in training the feature extractors (further details in a later section).

Storing the results:

We saved the networks giving the best results in each stage of building our model. These saved networks can be used to obtain predictions, evaluation on testing data and in further model tuning. Also, in this report we provided our training results and any further tuning made to obtain better results in each training stage.

The model:

In this section, we provide an abstract overview of the main parts in implementing and training the MRnet model. Further details and results are provided in the following sections.

Overview:

We had to slightly change the paper's approach due to a problem we faced. In the paper's implementation, transfer learning was applied to the feature extractor used (from ImageNet weights) so it will not require further training and can be used directly. To the contrary, it was required to build the feature extractors at

first from scratch without transfer learning and use them in building the full MRnet model. This required also training those extractors to extract the features correctly. This cannot be done by building the full MRnet building block as one coherent model as we faced a problem in the maximum pooling part. This is because the maximum pooling referred to in the paper's model is not done on each channel layer but on the contrary is done between the batches. In addition, it is applied to the output of a global average-pooling layer that gives a 2D tensor instead of a 3D one. So, we had to find a way to train the model bearing in mind that we cannot train the whole block at the same time due to the maximization operation separating between the feature extraction and the binary classification. We discussed three approaches to solve this problem, which are:

- Neglect the main MRnet model and build the network using only a single MRI in each series, so removing the maximization part entirely.
- Train the feature extractor using linear regression to try to give the same feature extraction as an extractor that has the same architecture but has the ImageNet weights.
- Combine the feature extractor with a binary classifier and train this network using only a single image from each series. Then the feature extractor can be obtained from this block and used in building the rest of the model.

We settled with the third approach. The first approach required completely changing the MRnet model by neglecting a major idea, which is doing the classification on the maximum values of each extracted feature from the s input images. The second approach may work, but it required training on the features extracted using a similar architecture having ImageNet weights which is far inferior to just using transfer learning, also the obtained features probably won't be specific to the required classification. The third approach, combined with data augmentation techniques seemed to be the logical way to go and most probably would give features specific to the classification problem at hand.

Model composition:

The model is composed of three main parts where each part is trained separately:

- A feature extraction network that is based on one of the required architectures to be implemented (VGG16, Resnet50, Inception V3). Having nine extractors for each architecture corresponding to the combinations of the three input series and the three output classifications.
- A binary classification network also based on the fully connected layers of each architecture. Also nine of those were required for each architecture.
- A logistic regression network to combine each three classifications of the same diagnosis into one. Three of which were required per architecture, one for each diagnosis.

Connecting the networks:

- Each extractor is connected to the corresponding binary classifier by passing the extractor output to a global average pooling layer then getting the maximum between the s output features to obtain as output a vector. This vector is passed as input to the classifier.
- The outputs of each three classifiers giving classification for the same diagnosis are combined in one array of shape 1x3 and given as input to the logistic regressor.
- The outputs of the three logistic regressors is the final output of the model.

Training the networks:

- To train each extractor, the extractor was attached to a fully connected layer and a binary classifier (different from the one previously mentioned). Then this network is trained using a single image as input from each series. Here we used data augmentation to increase the input size and obtain better results (considering that the usage of one image instead of s from each exam decreases the input size considerably). The data augmentation technique used was based on the same method used in the paper.
- Having trained the extractor, it can be used to extract features of the input giving an output that is passed to global average and max layers to get the input vector of the classifier. This vector along with the corresponding output classification is used to train the binary classifier of our model.
- Having trained the classifiers, the classifications of each three classifiers that classify the same diagnosis can be used as input for the logistic

regressor. This input alongside the actual output classification was used in training the regressor. Which completes the training of the whole model.

- During the training of each block, two callbacks were used one to save the model giving the best validation accuracy and one for early stopping on the validation loss with patience of 5 epochs.
- A Batch size of 20 and a total number of epochs of 50 (with early stopping) was used.

Testing the model:

- Each extractor was tested using a single image from the test input along with the classification (notice that the testing is on the extractor after combining it with the binary classifier as done in training).
- Each binary classifier was tested using the input as the features of the test input extracted using the corresponding feature extractor and using the corresponding classifications as test outputs.
- Each regressor was tested having the corresponding three classifications as input (from the corresponding binary classifiers) for each diagnosis from the three input series as its input. The test output was the corresponding classifications.
- By this approach, we can say that the final test on the regressor is a test on the complete model, as it takes its inputs the predictions of the other networks (extractors and classifiers) in the model.

Obtaining the predictions:

To obtain the predictions of the model based on the three input series of an exam the following steps are made:

- First, the image series are passed to the corresponding extractor to extract the features, one for each of the series-diagnosis pair, so nine total.
- Each of the extracted features is passed to the corresponding binary classifier to give a probability of having the diagnosis according to the given extracted features, also nine total.
- Each three classifications for the same diagnosis is then passed to the corresponding logistic regressor to perform a weighted sum and obtain a final diagnosis probability, three total outputs.

- The three outputs of the regressor are considered the final output of the model.

Note that at this stage, all the feature extractors, binary classifiers and logistic regressors are trained by the techniques specified.

The implementation of the functions manipulating the model parts was done generically to allow for the use of different architectures to implement any part of the network.