Name: Sherif Mohamed Mostafa ID:22

"Individual work report"

Overview of my work:

In this report, I am going to clarify my contribution and impact in the project. My role mainly focused on dealing with the data set, dealing with training the model (the general flow of the process), performing transfer learning and helping in the contribution. This includes:

- Summarizing the MRnet paper.
- Understanding the problem and the data set to help clarify it to the team.
- Building the functions used in training and testing the model (based on the approach that we agreed upon, to be shown in details in a later section).
- Performing transfer learning using architectures based on both inception V3 and VGG16 (ResNet was also tried out by a teammate).
- Making the feature extractor code for one of the two models used in the contribution part.

These are the notebooks having my implementation and results:

- MRnet_models_processing
- InceptionV3_transfer_learning
- VGG16_transfer_learning

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.
- Data augmentation was used in training the feature extractors (further details in a later section).
- Obtaining the data in batches wasn't needed according to the flow we followed that only required a single set of training images at a time which could fit in the RAM.

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. In addition, in this report I provided some testing results and any further tuning made to obtain better results in the training stage. For the whole collection of training outputs, please refer to the provided colab notebooks.

The model:

In this section, I provide an abstract overview of the main parts in implementing and training the MRnet model. I also provide the results when using this approach with transfer learning. Further details and results are provided in the notebooks and other reports.

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. We could not do this 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 slices. In addition, it is applied to the output of a global average-pooling layer that gives a 2D tensor instead of a 3D one. Therefore, 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 will not 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 could be 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 that could also be 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 (logistic regression based) 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.

Link of notebook containing the implementation in colab:

MRnet_models_processing.

Transfer learning:

In this part I performed transfer learning using Inception V3 and VGG16. **Note that the required was to do the transfer learning on the model giving the best results, but the results came out to be fairly close so we tried transfer learning to see if any significant difference would occur in the results. We found that ResNet50 gave the best results when using transfer learning.** When comparing the results of the different approaches in the transfer learning, I show the graph for the **axial-abnormal classification.**

The impact of transfer learning:

It is a commonly used approach in deep learning algorithms to tackle the problem of having small data sets and wanting to achieve better results. It is because a CNN used in deep learning simply extracts certain features from the input to be then used in classifying the input or in further processing. Therefore, it is

understood that similar problems would benefit from the same features; in this case, using a pre-trained network on a similar problem with its learnt weights is a good approach to tackling the problem at hand.

The pre-trained networks used:

We used CNN networks having weights initialized to the results of training on the 'ImageNet' data set. This is because it is similar to our problem where both are image classification problems so the features obtained in both cases are similar to a large extent.

The ImageNet data set:

Image net is a large database of labeled images, organized in a tree-like hierarchy according to the WordNet hierarchy. Now image net has 14,197,122 images and average of 500 images per node. Image net dataset was used over several years to test new CNN architectures capabilities in image recognition and classification capabilities. These architectures include VGG16, ResNet and Inception V3.

Reference for image net dataset: http://www.image-net.org/

Results of using InceptionV3 with transfer learning:

The first architecture I tried in transfer learning was using the inceptionV3 built in Keras.

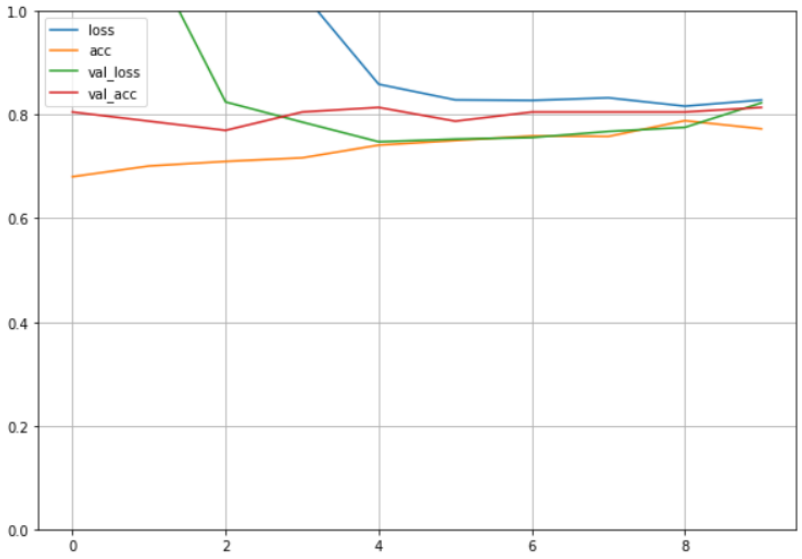
1. ### The used inception model:
    - Inception V3 was used with a binary-cross entropy loss function, 10^-4 learning rate and adam optimizer.
    - The extractor was used directly without further training having weights transferred from image net.
2. ### The binary classifier used:
    - In this part I tried to find the best classifier parameters giving good performance without over fitting.
    - First, I tried a simple classifier model having two dense layers of 128 and 64 nodes with relu activation. L2 regularization also added and dropout layers with 50% probability of dropout. A final output layer was added having sigmoid activation for binary classification. Similar optimizer and learning rate was used as that used with the feature extractor.

- First classifier summary:

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 128)               262272
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 270,593
Trainable params: 270,593
Non-trainable params: 0
```

- The output of this classifier was under fitting in its loss, which showed that a more complex classifier may be needed.

```
51/51 [==============================] - 0s 5ms/step - loss: 0.8285 - acc: 0.7729 - val_loss: 0.8230 - val_acc: 0.8142
Epoch 00010: early stopping
```

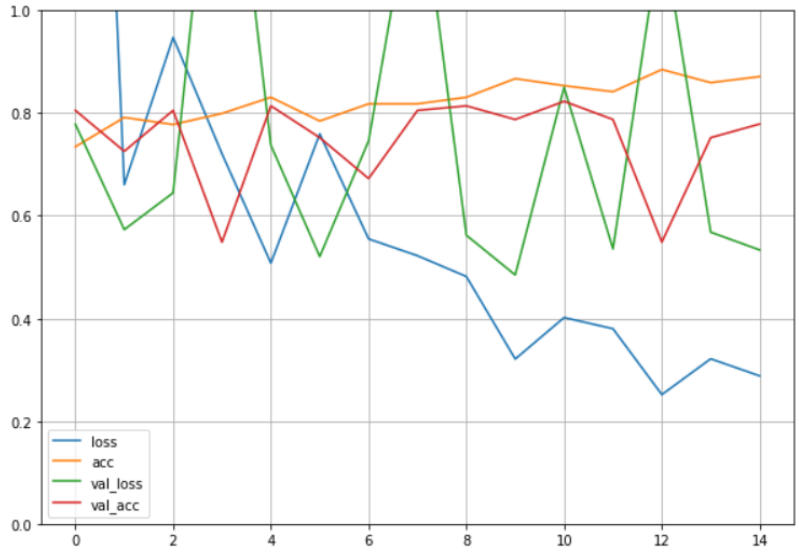


- Second, I tried a more complex classifier format by changing the number of the neurons in the 2 dense layers to 1024 and 512. I also removed the dropout and regularization to test the effect of doing so.
- Second classifier summary:

```
Model: "sequential_40"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_33 (Dense)             (None, 1024)              2098176
_________________________________________________________________
dense_34 (Dense)             (None, 512)               524800
_________________________________________________________________
dense_35 (Dense)             (None, 1)                 513
=================================================================
Total params: 2,623,489
Trainable params: 2,623,489
Non-trainable params: 0
_________________________________________________________________
```

- The output of this classifier gave lower loss but had several oscillations and was clearly overfitting.

```
51/51 [==============================] - 0s 5ms/step - loss: 0.2881 - acc: 0.8712 - val_loss: 0.5335 - val_acc: 0.7788
Epoch 00015: early stopping
```

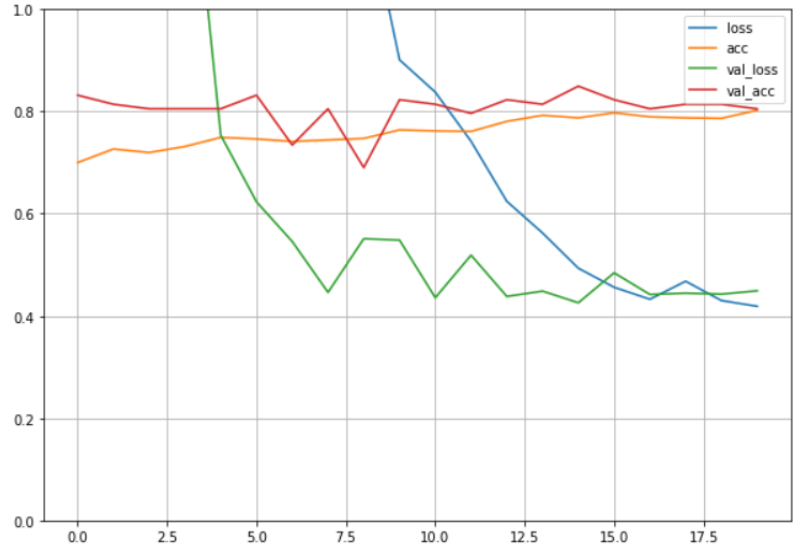


- Third, I tried using the previous classifier but also adding dropout layers seemingly gave the best results. It eventually converged to a low loss value and over fitting then was negligible.
- Third classifier summary:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1024)              2098176
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 2,623,489
Trainable params: 2,623,489
Non-trainable params: 0
_________________________________________________________________
```

- Third classifier output, somewhat satisfactory. This classifier was the one used finally.

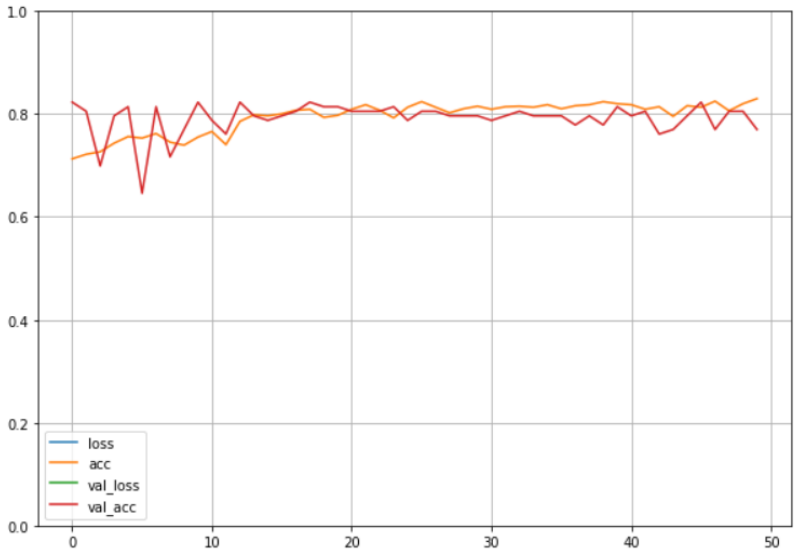
```
51/51 [==============================] - 0s 6ms/step - loss: 0.4189 - acc: 0.8024 - val_loss: 0.4492 - val_acc: 0.8053
Epoch 00020: early stopping
```



- Here is a look at some other trials that gave no better performance:
    - Adding L2 regularization to the final classifier, this lead to a large increase in the loss. Note that the graph is drawn at a range from 0 to 1. This means that here the loss never decreased bellow 1.

```
Model: "sequential_37"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_30 (Dense)             (None, 1024)              2098176
_________________________________________________________________
dropout_20 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_31 (Dense)             (None, 512)               524800
_________________________________________________________________
dropout_21 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_32 (Dense)             (None, 1)                 513
=================================================================
Total params: 2,623,489
Trainable params: 2,623,489
Non-trainable params: 0
_________________________________________________________________
```

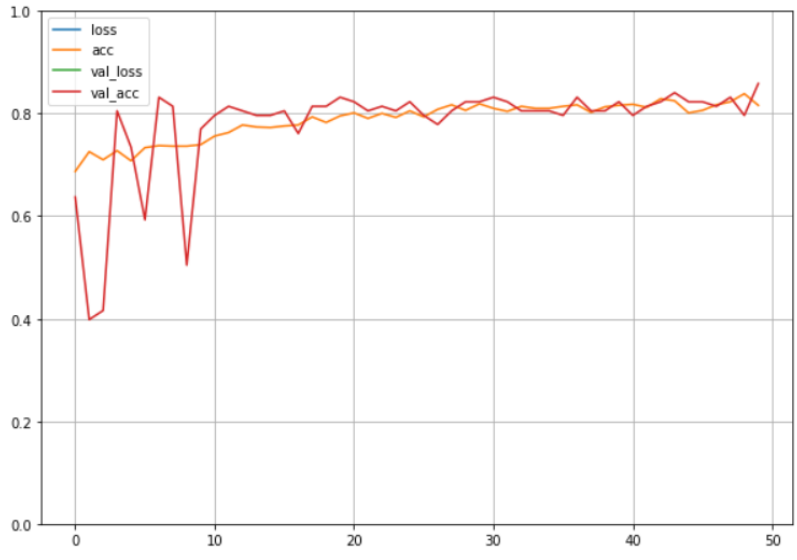
```
Epoch 00050: val_acc did not improve from 0.82301
51/51 [==============================] - 0s 6ms/step - loss: 1.5092 - acc: 0.8299 - val_loss: 1.5866 - val_acc: 0.7699
```



- Trying a more complex classifier as an effort to improve the output. This was done by adding an additional dense layer with 256 neurons. This modification was done to the previous classifier, the one using the L2 regularization. The loss was high still, and apparent oscillations.

```
Model: "sequential_42"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_36 (Dense)             (None, 1024)              2098176
_________________________________________________________________
dropout_24 (Dropout)         (None, 1024)              0
_________________________________________________________________
dense_37 (Dense)             (None, 512)               524800
_________________________________________________________________
dropout_25 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_38 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_39 (Dense)             (None, 1)                 257
=================================================================
Total params: 2,754,561
Trainable params: 2,754,561
Non-trainable params: 0
_________________________________________________________________
```

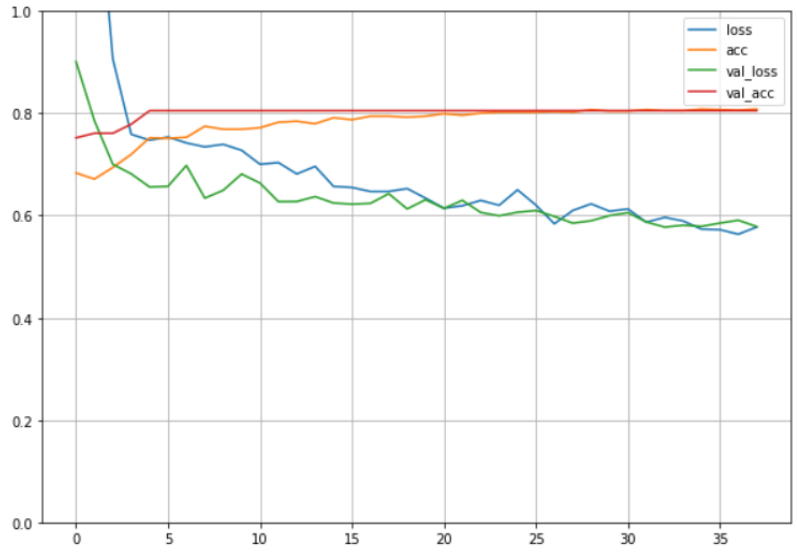
```
Epoch 00050: val_acc improved from 0.84071 to 0.85841, saving model to /content/drive/My Drive/Models/InceptionV3_transf
51/51 [==============================] - 1s 11ms/step - loss: 1.8067 - acc: 0.8161 - val_loss: 1.8170 - val_acc: 0.8584
```



- When trying to use less number of neurons per layer, the model became so simple and the accuracy obtained was lower than before. However, the oscillation and high loss problems were resolved.

```
Model: "sequential_47"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_44 (Dense)             (None, 64)                131136
_________________________________________________________________
dropout_28 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_45 (Dense)             (None, 32)                2080
_________________________________________________________________
dropout_29 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_46 (Dense)             (None, 16)                528
_________________________________________________________________
dense_47 (Dense)             (None, 1)                 17
=================================================================
Total params: 133,761
Trainable params: 133,761
Non-trainable params: 0
_________________________________________________________________
```

```
51/51 [==============================] - 0s 5ms/step - loss: 0.5777 - acc: 0.8083 - val_loss: 0.5786 - val_acc: 0.8053
Epoch 00038: early stopping
```

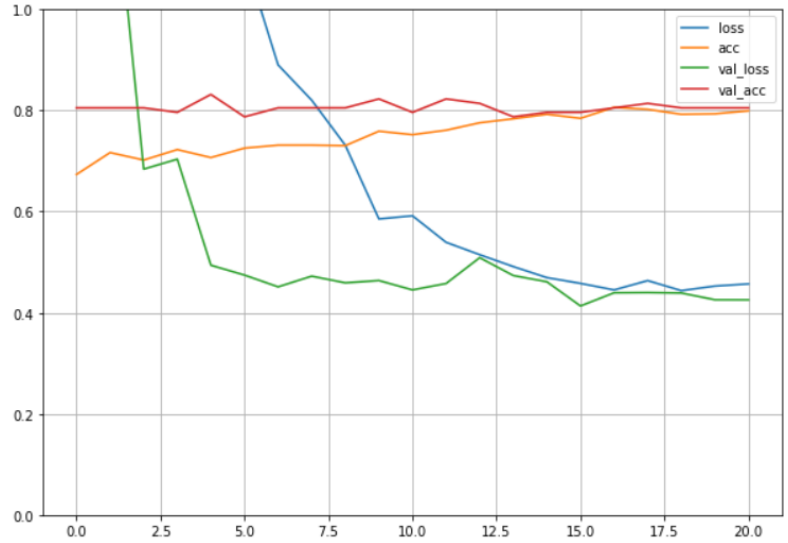


Model: "sequential_57"

Layer (type)	Output Shape	Param #
dense_60 (Dense)	(None, 512)	1049088
dropout_36 (Dropout)	(None, 512)	0
dense_61 (Dense)	(None, 256)	131328
dropout_37 (Dropout)	(None, 256)	0
dense_62 (Dense)	(None, 128)	32896
dense_63 (Dense)	(None, 1)	129

Total params: 1,213,441
Trainable params: 1,213,441
Non-trainable params: 0

51/51 [==============================] - 0s 5ms/step - loss: 0.4570 - acc: 0.7994 - val_loss: 0.4255 - val_acc: 0.8053
Epoch 00021: early stopping



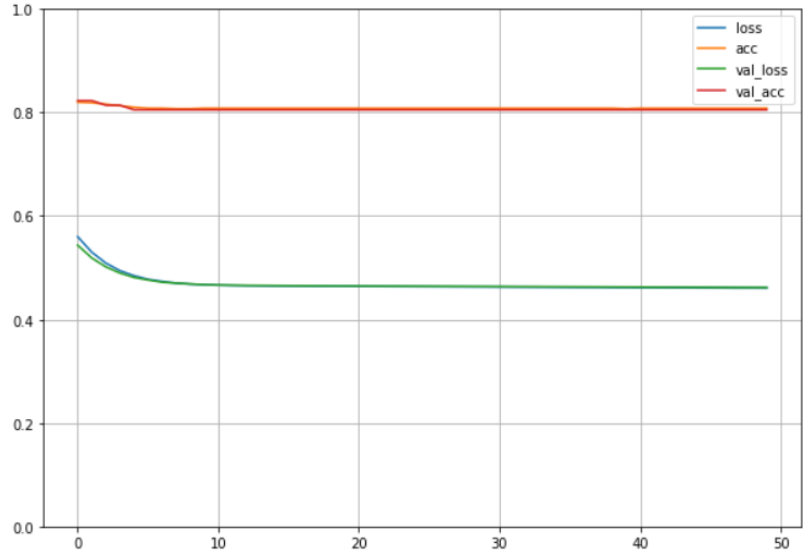
3. The logistic regression:

- The regressor used was a simple one dense layer of 1 neuron, activation sigmoid, loss binary-cross entropy and optimizer adam with a learning rate of 10^-4.
- The simple architecture is because the input consists of only 3 numbers and the task is simple logistic regression.
- Regressor summary:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 1)                 4
=================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
_________________________________________________________________
```

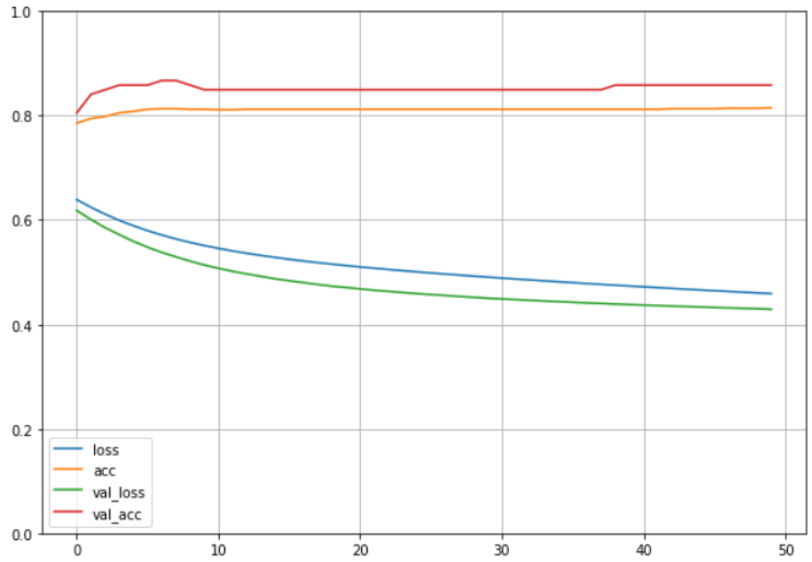
- Results of training the three regressors:
    1. Abnormal:

```
51/51 [==============================] - 0s 3ms/step - loss: 0.4608 - acc: 0.8083 - val_loss: 0.4618 - val_acc: 0.8053
```



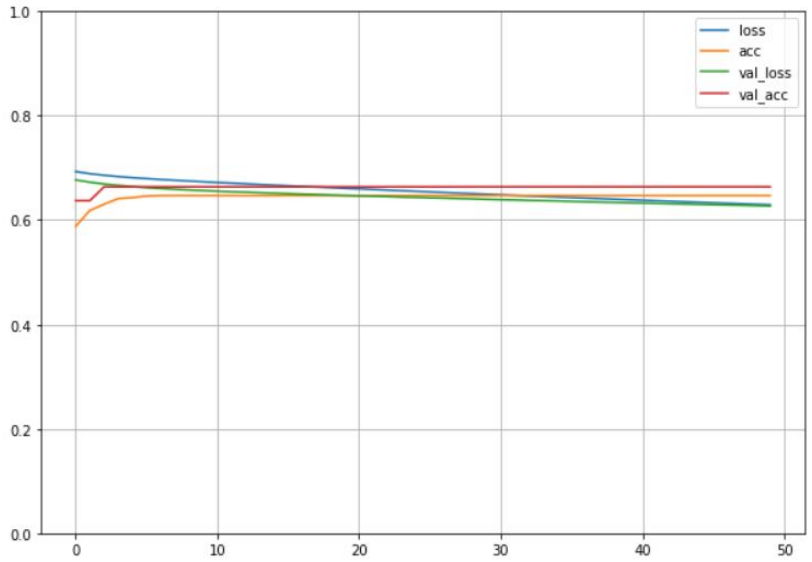
2. ACL:

```
51/51 [==============================] - 0s 3ms/step - loss: 0.4590 - acc: 0.8151 - val_loss: 0.4291 - val_acc: 0.8584
```



3. Meniscal:

```
51/51 [==============================] - 0s 3ms/step - loss: 0.6293 - acc: 0.6470 - val_loss: 0.6270 - val_acc: 0.6637
```



4. The final test results:

Anomaly	Accuracy	Loss
Abnormal	79.17%	0.5504
ACL	54.17%	0.7141
Meniscal	56.67%	0.7

5. Notebook link for the complete work:

InceptionV3_transfer_learning

Results of using VGG16 with transfer learning:

The second architecture I tried in transfer learning was using the VGG16 built in Keras.

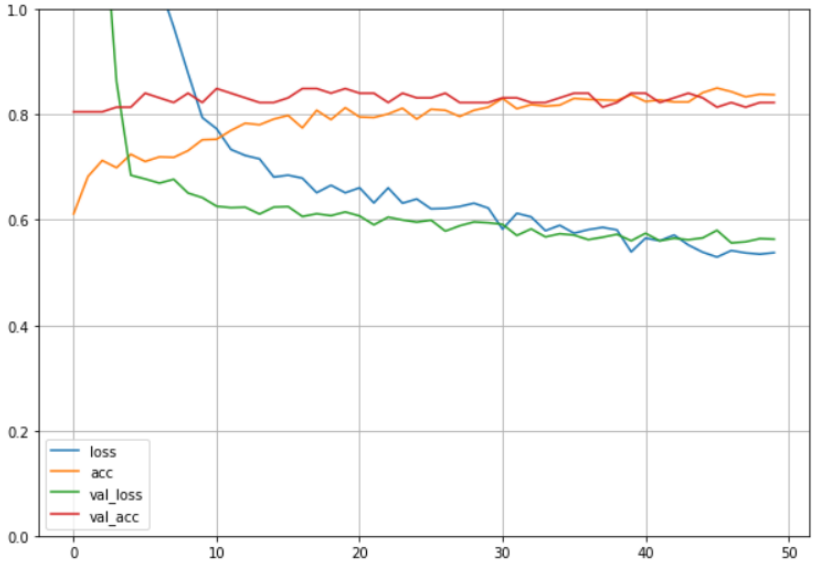
1. The used VGG16 model:
   - VGG16 was used with a binary-cross entropy loss function, 10^-4 learning rate and adam optimizer.
   - The extractor was used directly without further training having weights transferred from image net.
2. The binary classifier used:
   - First, I tried a simple classifier model having two dense layers of 128 and 64 nodes with relu activation. L2 regularization also added and dropout layers with 50% probability of dropout. A final output layer was added having sigmoid activation for binary classification. Similar optimizer and learning rate was used as that used with the feature extractor. This is the first classifier used in the inception V3 transfer learning part too.

- First classifier summary:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 128)               65664
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 73,985
Trainable params: 73,985
Non-trainable params: 0
_________________________________________________________________
```

- The output of this classifier was under fitting in its loss, which showed that a more complex classifier may be needed. Although here the under fitting is much less than the case of inception V3.

```
Epoch 00050: val_acc did not improve from 0.84956
51/51 [==============================] - 0s 3ms/step - loss: 0.5377 - acc: 0.8378 - val_loss: 0.5634 - val_acc: 0.823
```

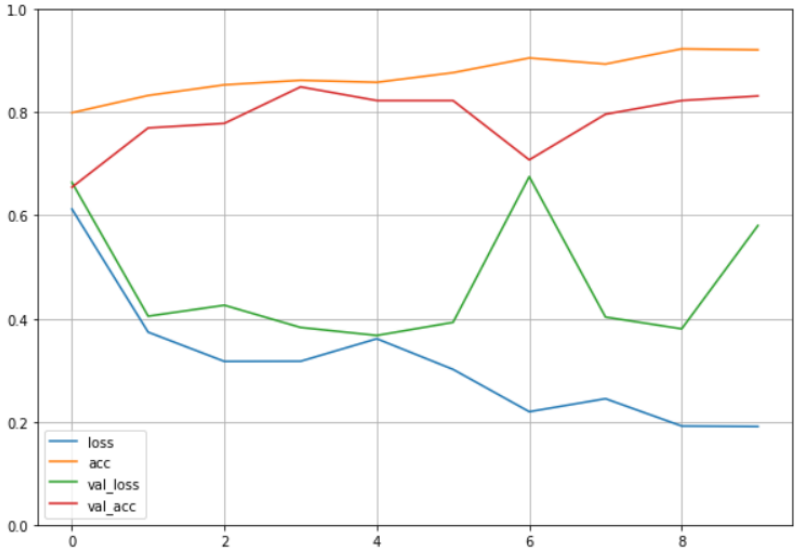


- Second, I tried a more complex classifier format by changing the number of the neurons in the 2 dense layers to 1024 and 512. I also removed the dropout and regularization to test the effect of doing so. Same approach used in inception V3.
- Second classifier summary:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 1024)              525312
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 1,050,625
Trainable params: 1,050,625
Non-trainable params: 0
_________________________________________________________________
```

- The output of this classifier gave lower loss but had several oscillations and was clearly over fitting. Similar to what happened in the inception V3 case.

```
51/51 [==============================] - 0s 3ms/step - loss: 0.1905 - acc: 0.9213 - val_loss: 0.5804 - val_acc: 0.8319
Epoch 00010: early stopping
```

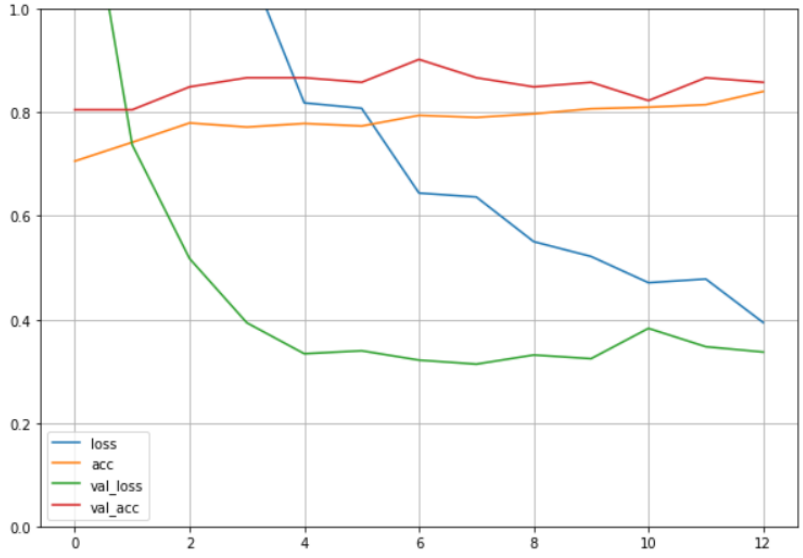


- Third, I tried using the previous classifier but also adding dropout layers seemingly gave the best results. It eventually converged to a low loss value and over fitting then was negligible. It gave better-looking results than the inception V3 case.
- Third classifier summary:

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 1024)              525312
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 513
=================================================================
Total params: 1,050,625
Trainable params: 1,050,625
Non-trainable params: 0
_________________________________________________________________
```

- Third classifier output, somewhat satisfactory. This classifier was the one used finally.

```
51/51 [==============================] - 0s 5ms/step - loss: 0.3940 - acc: 0.8407 - val_loss: 0.3369 - val_acc: 0.8584
Epoch 00013: early stopping
```



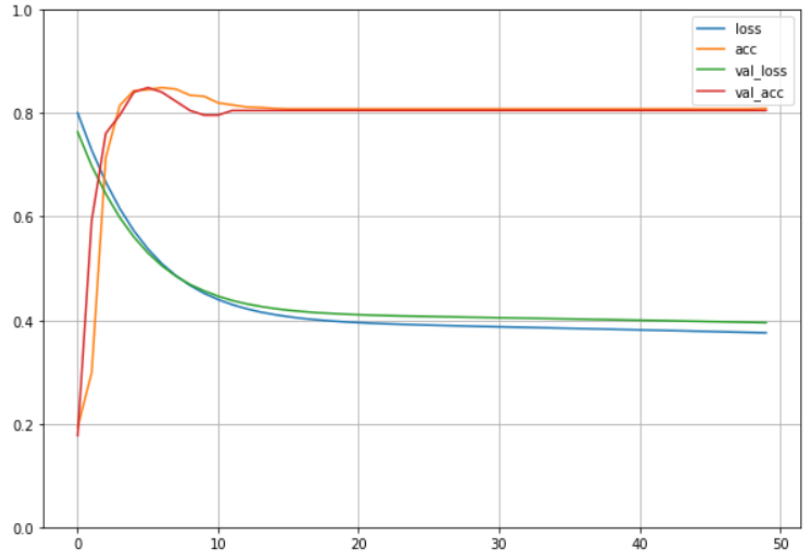
3. The logistic regression:

- The regressor used was a simple one dense layer of 1 neuron, activation sigmoid, loss binary-cross entropy and optimizer adam with a learning rate of 10^-4. Same as the network used in the inception V3 case.
- The simple architecture is because the input consists of only 3 numbers and the task is simple logistic regression.
- Regressor summary:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 1)                 4
=================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
_________________________________________________________________
```

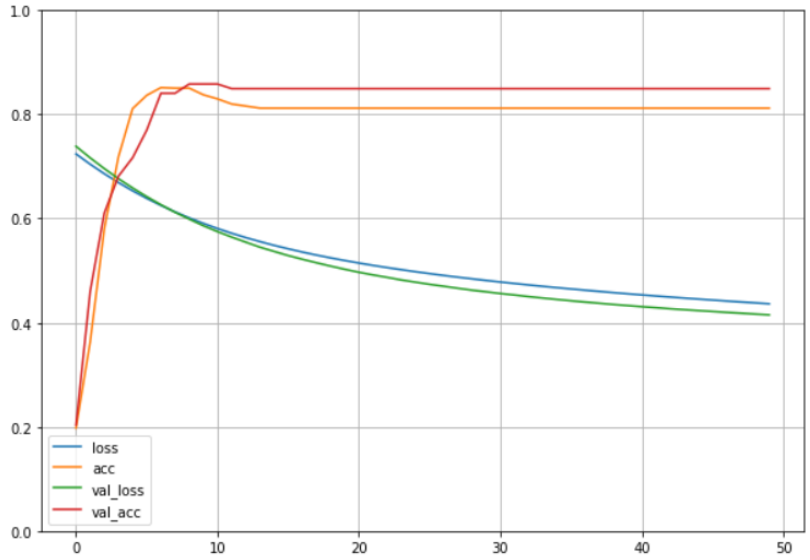
- Results of training the three regressors:
    i. Abnormal:

```
51/51 [==============================] - 0s 3ms/step - loss: 0.3756 - acc: 0.8092 - val_loss: 0.3952 - val_acc: 0.8053
```



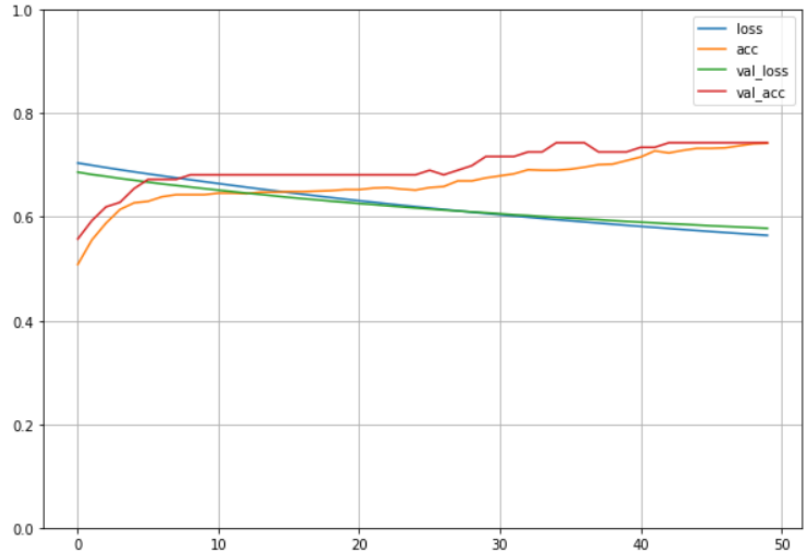
ii. ACL:

```
51/51 [==============================] - 0s 3ms/step - loss: 0.4364 - acc: 0.8122 - val_loss: 0.4151 - val_acc: 0.8496
```



iii. Meniscal:

```
51/51 [==============================] - 0s 3ms/step - loss: 0.5645 - acc: 0.7424 - val_loss: 0.5781 - val_acc: 0.7434
```



4. The final test results:

The results were found to be better than using inception V3 with the same binary classifier architecture.

Anomaly	Accuracy	Loss
Abnormal	80.83%	0.5338
ACL	62.5%	0.6529
Meniscal	59.17%	0.6374

5. Notebook link for the complete work:

VGG16_transfer_learning

My work in the contribution part:

- In this part I added the implementation of a CNN feature extractor model based on the paper titled *"Using Deep Learning Algorithms to Automatically Identify the Brain MRI Contrast: Implications for Managing Large Databases"*. The main architecture is based on CNN layers of 3x3 filters where the extractor consisits of 3 similar blocks:
- The first block has 32 filters in convolutional layers, the second has 64 and the third has 128.
- Each block has 2 convolutional layers separated by batch normalizations and a relu activation layer.
- Between each 2 blocks there is a max pooling layer of size 2x2.

- Extractor summary:

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_19 (Conv2D)           (None, 256, 256, 32)      896
_________________________________________________________________
batch_normalization_18 (Batc (None, 256, 256, 32)      1024
_________________________________________________________________
activation_18 (Activation)   (None, 256, 256, 32)      0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 254, 254, 32)      9248
_________________________________________________________________
batch_normalization_19 (Batc (None, 254, 254, 32)      1016
_________________________________________________________________
activation_19 (Activation)   (None, 254, 254, 32)      0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 127, 127, 32)      0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 127, 127, 64)      18496
_________________________________________________________________
batch_normalization_20 (Batc (None, 127, 127, 64)      508
_________________________________________________________________
activation_20 (Activation)   (None, 127, 127, 64)      0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 125, 125, 64)      36928
_________________________________________________________________
batch_normalization_21 (Batc (None, 125, 125, 64)      500
_________________________________________________________________
activation_21 (Activation)   (None, 125, 125, 64)      0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 62, 62, 64)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 62, 62, 128)       73856
_________________________________________________________________
batch_normalization_22 (Batc (None, 62, 62, 128)       248
_________________________________________________________________
activation_22 (Activation)   (None, 62, 62, 128)       0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 60, 60, 128)       147584
_________________________________________________________________
batch_normalization_23 (Batc (None, 60, 60, 128)       240
_________________________________________________________________
activation_23 (Activation)   (None, 60, 60, 128)       0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 30, 30, 128)       0
=================================================================
Total params: 290,544
Trainable params: 288,776
Non-trainable params: 1,768
_________________________________________________________________
```

Main Results of my work:

- Establishing the notebook later used in all training and testing operations.
- Finding the results of using VGG16 and inception V3 with transfer learning and comparing them.

Problems faced:

- I was responsible for dealing with the data and building the model framework. I faced a problem as was stated in the maximum pooling part of the model. Despite several attempts I could not find a method to solve it and the training method had to be ready to start training as soon as possible, so we discussed the solutions stated as a way to handle this problem.
- Data normalization was implemented and used in training, however after completing the training; I revised the code quickly and found that although variables were being normalized in the function, they were not stored in the matrix containing the input, as it should be. This was discovered at a late timing and there was no time to roll back to restart the training, so we removed the normalization part from the training function as it is meaningless anyways. However, the normalization function itself is present in the notebook MRnet_models_processing.