OPERATION RESEARCH

Individual report(2)

Name : Hazem Morsy Hassan

ID : 18

Summary and notes on the work done:

- Implementing VGG model from scratch using keras
- Training extractors , classifiers and regressors of VGG model
- Implementation and training of RESNet model for transfer learning (classifiers and regressors)
- Summarizing the first paper in the contribution part and training the new model mentioned in it (DenseNet) by transfer learning.

All training trials are done axial-abnormal extractors and classifiers and on abnormal regressors except one test for a classifier in resnet transfer learning model is done on Axial ACL and it will be mentioned again later.

VGG16 model :

I started by building the VGG model from scratch which is used for training the extractors to identify the features of each anomaly(abnormal , ACL and Meniscal) according to each series (Axial , Sagittal and Coronal). The model is simple due to using simple hyperparameters in which all the filters of convolutional layers are of the same size (3 X 3) and same padding and filters of the max-pooling layers are of size (2 X 2) and stride of 2

Model structure:

- Two convolutional layers of 3 x 3 kernel with 64 channels and same padding
- Max pool layer of 2 X 2 pool size and stride of 2 X 2
- Two convolutional layers of 3 x 3 kernel with 128 channels and same padding
- Max pool layer of 2 X 2 pool size and stride of 2 X 2
- Dropout layer of probability (0.5)
- Three convolutional layers of 3 x 3 kernel with 256 channels and same padding
- Max pool layer of 2 X 2 pool size and stride of 2 X 2
- Three convolutional layers of 3 x 3 kernel with 512 channels and same padding
- Max pool layer of 2 X 2 pool size and stride of 2 X 2
- Dropout layer of probability (0.5)

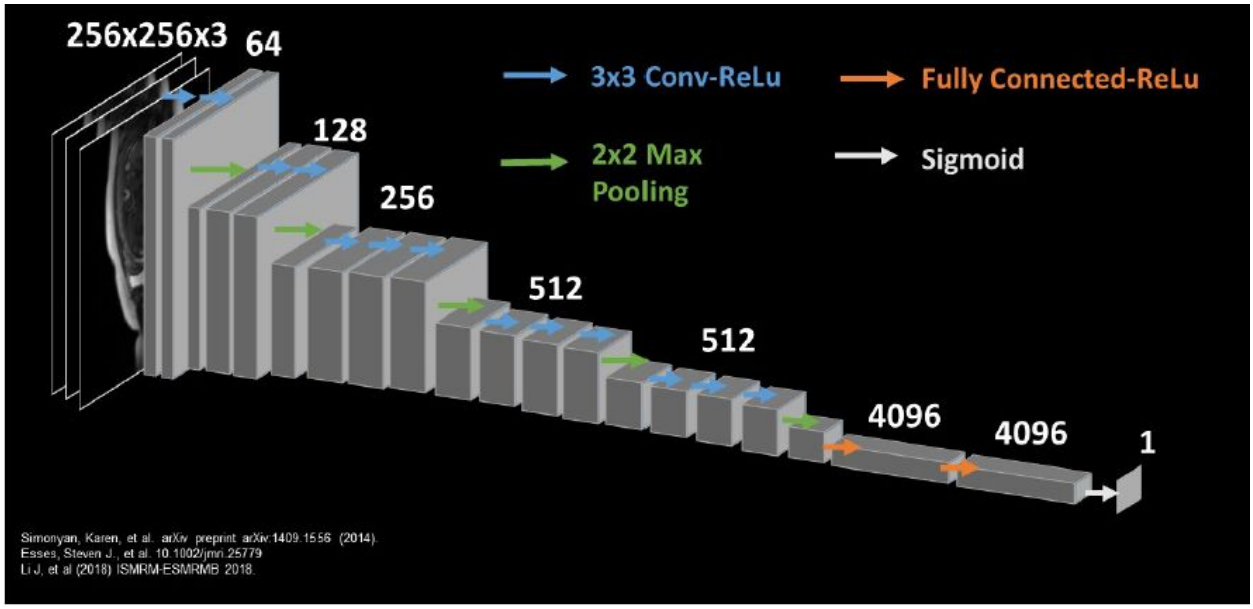
- Three convolutional layers of 3 x 3 kernel with 512 channels and same padding
- Max pool layer of 2 X 2 pool size and stride of 2 X 2
- Dense layer of 4096 neurons
- Dropout layer of probability (0.5)
- Dense layer of 4096 neurons
- Dropout layer of probability (0.5)
- Dense layer of 1 neuron with sigmoid function

VGG Model Summary:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 64)      1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 256, 64)      36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 128)     73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 128, 128, 128)     147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 128)       0
_________________________________________________________________
dropout (Dropout)            (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 256)       295168
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 256)       590080
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 64, 64, 256)       590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 512)       1180160
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 512)       2359808
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 32, 32, 512)       2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 512)       0
_________________________________________________________________
```

dropout_1 (Dropout)	(None, 16, 16, 512)	0
conv2d_10 (Conv2D)	(None, 16, 16, 512)	2359808
conv2d_11 (Conv2D)	(None, 16, 16, 512)	2359808
conv2d_12 (Conv2D)	(None, 16, 16, 512)	2359808
max_pooling2d_4 (MaxPooling2	(None, 8, 8, 512)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 4096)	134221824
dropout_2 (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312
dropout_3 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 1)	4097

Total params: 165,721,921
Trainable params: 165,721,921
Non-trainable params: 0



Model Compile:

Optimizer used : ADAM

Learning rate : 10^-4

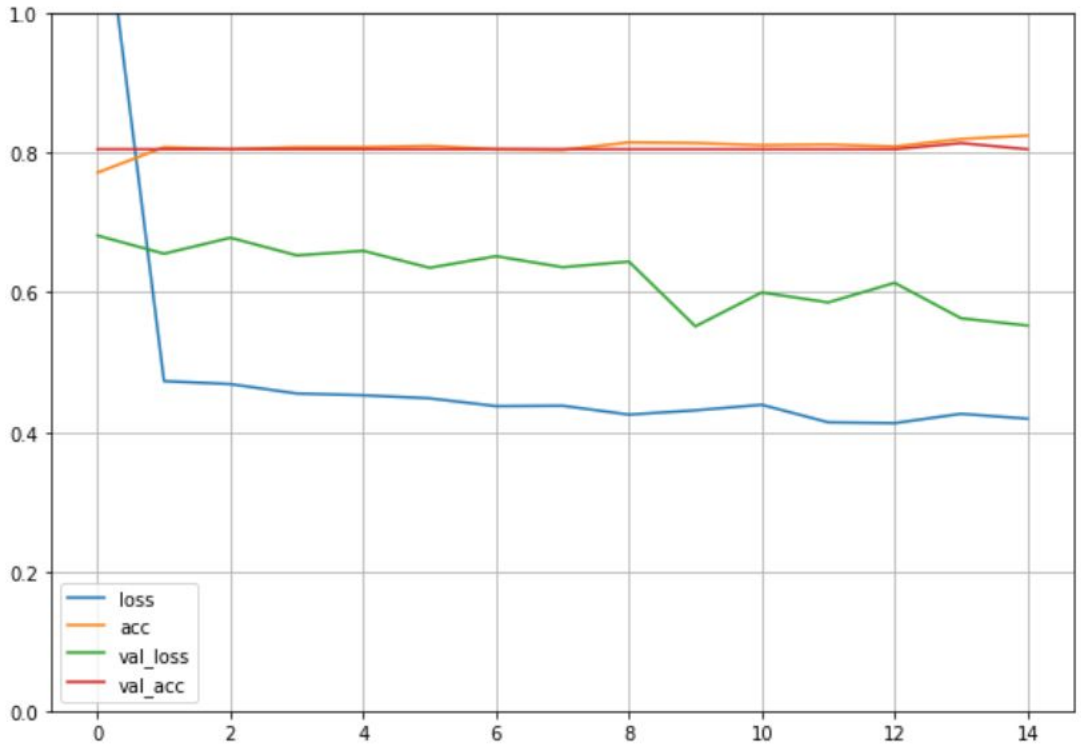
Loss function : binary cross entropy for classification

Notes:

- At first I trained the full model of VGG including the classifier for one slice only for each anomaly and according to each series.
- Then I use the pop function to remove the last six layers (from flatten layer until the dense layer of 1 neuron) form the trained model to become only a feature extractor which stops at the last max pooling layer of output (8 X 8 X 512)

Training Extractor Trials :

- Using many dropout layers (use dropout after each maxpool layer)

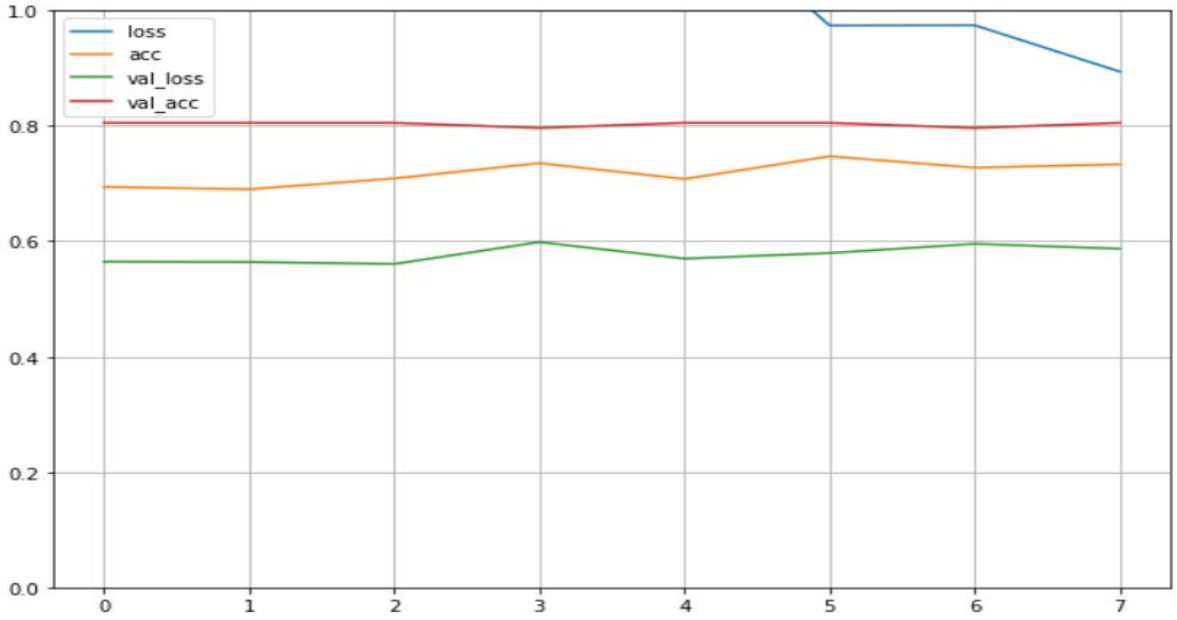



```
<tensorflow.python.keras.callbacks.History at 0x7f7eb02f29e8>
```

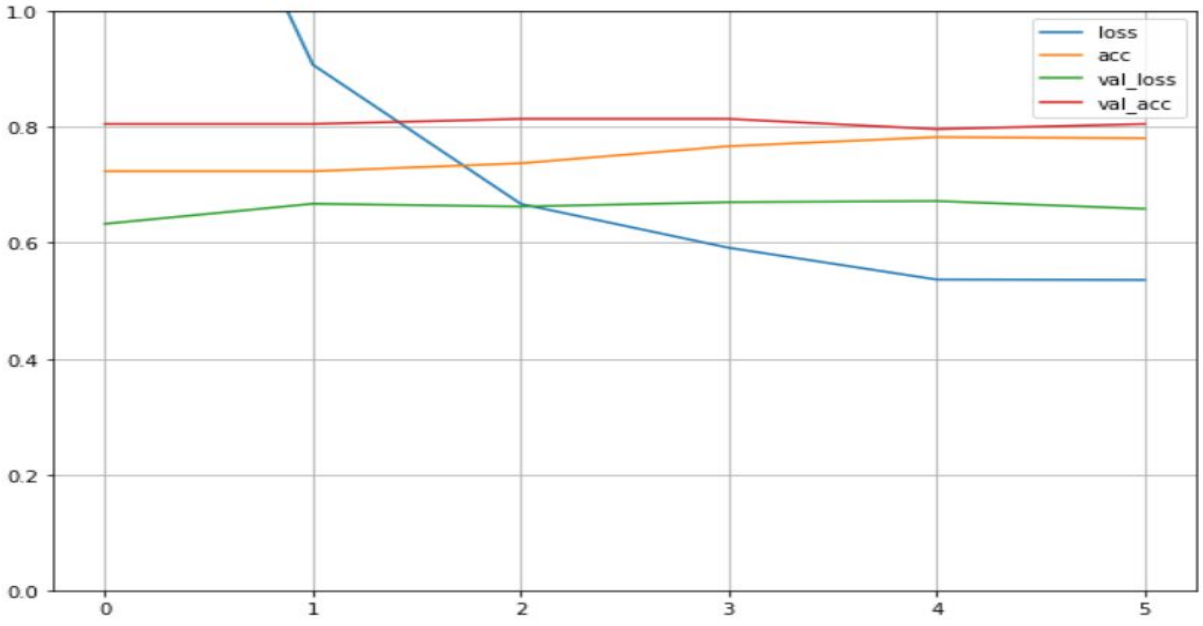
```
model_temp =  load_model(data.vgg,data.axial,data.abnormal,data.extractor)
list1 = test_extractor(model_temp,data.axial,data.abnormal)
```

```
4/4 [==============================] - 1s 210ms/step - loss: 1.9590 - acc: 0.8000
```

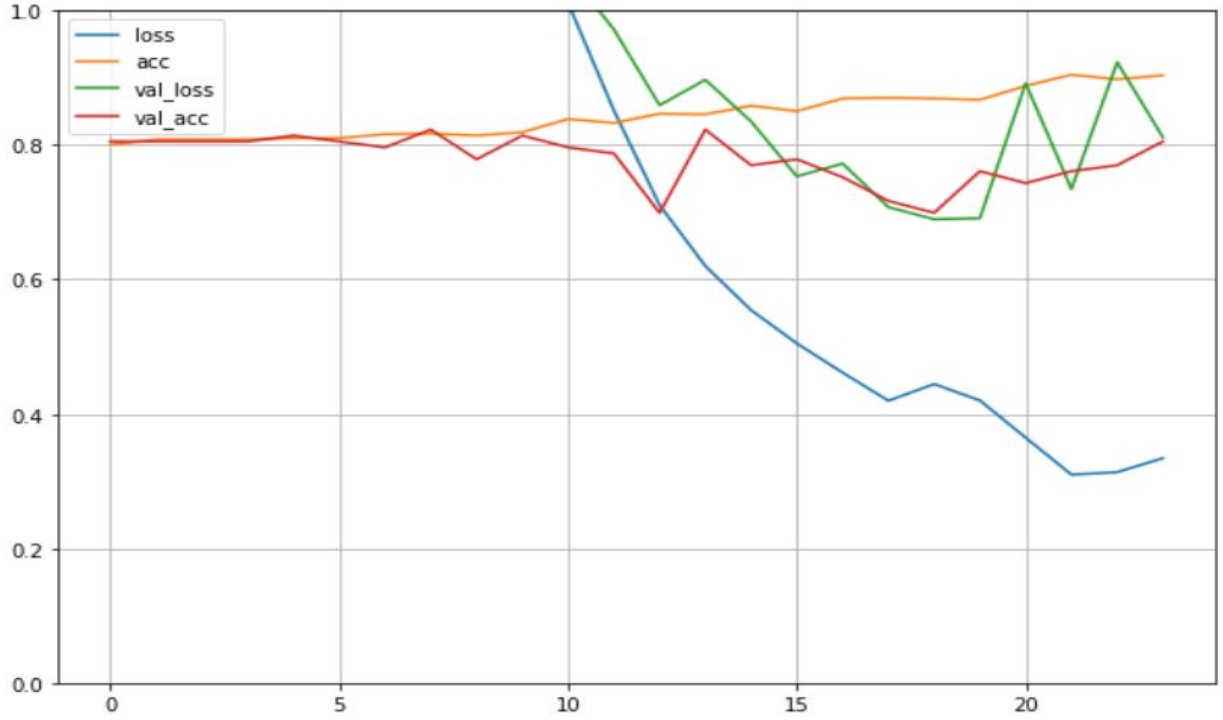
- Decreasing learning rate to be 10^-6 but that makes loss decades slowly



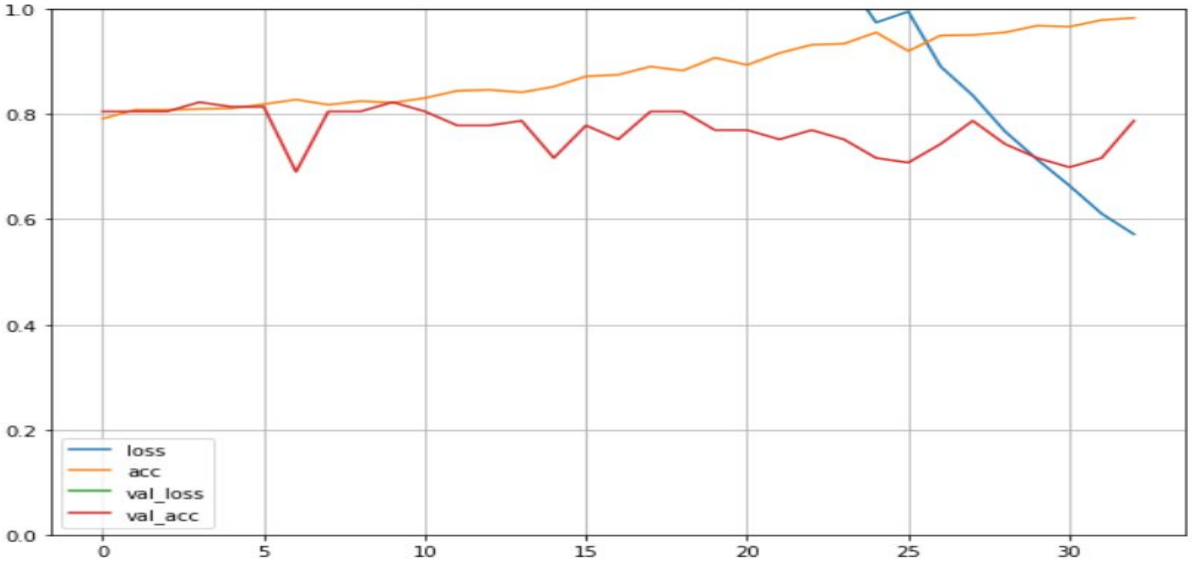
- Decreasing learning rate to be 10^-5 but that makes loss decades slowly



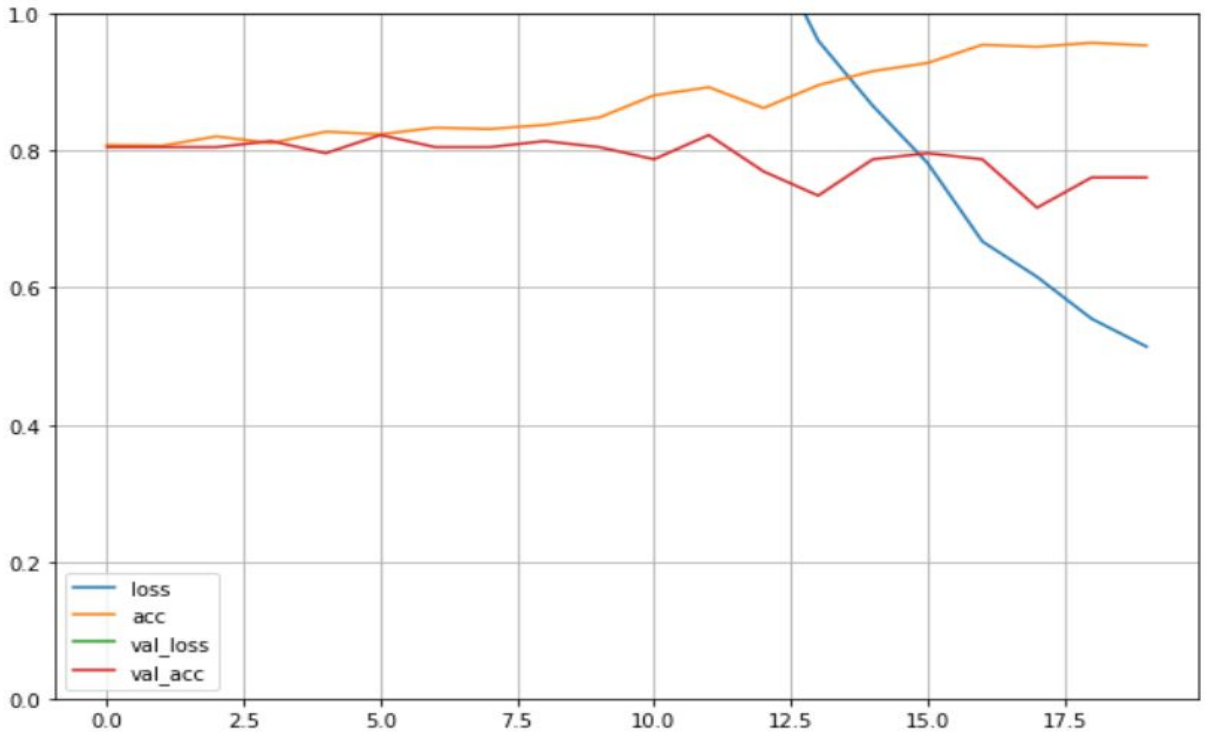
- Use regularization term = 0.01 but there still overfitting



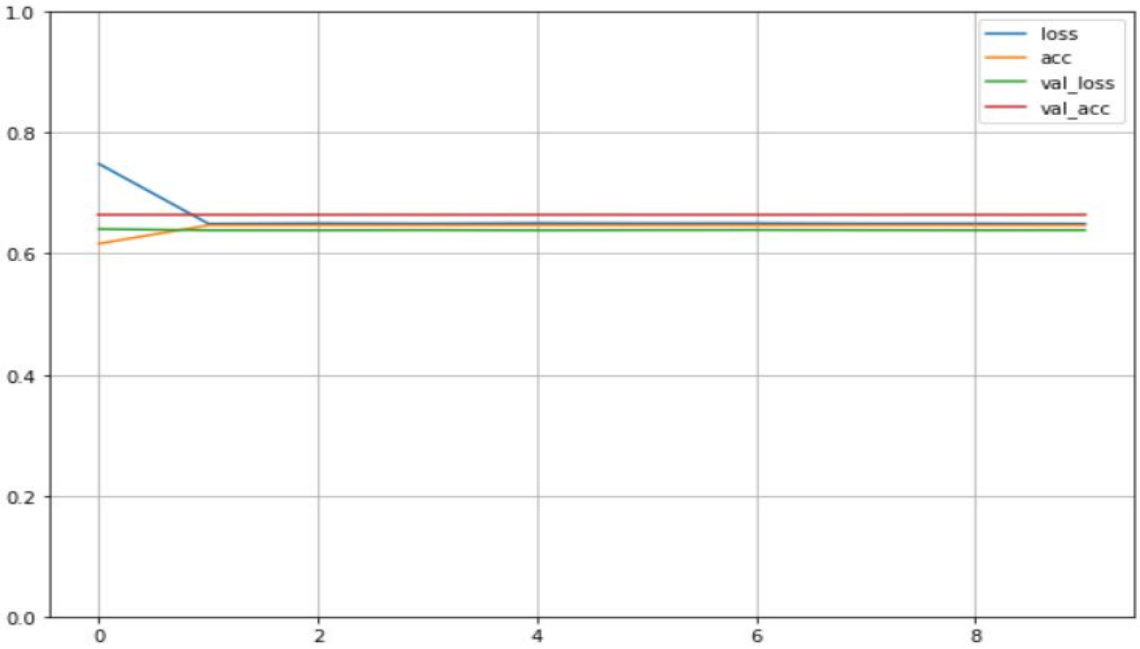
- Use regularization term = 0.001 causes high loss



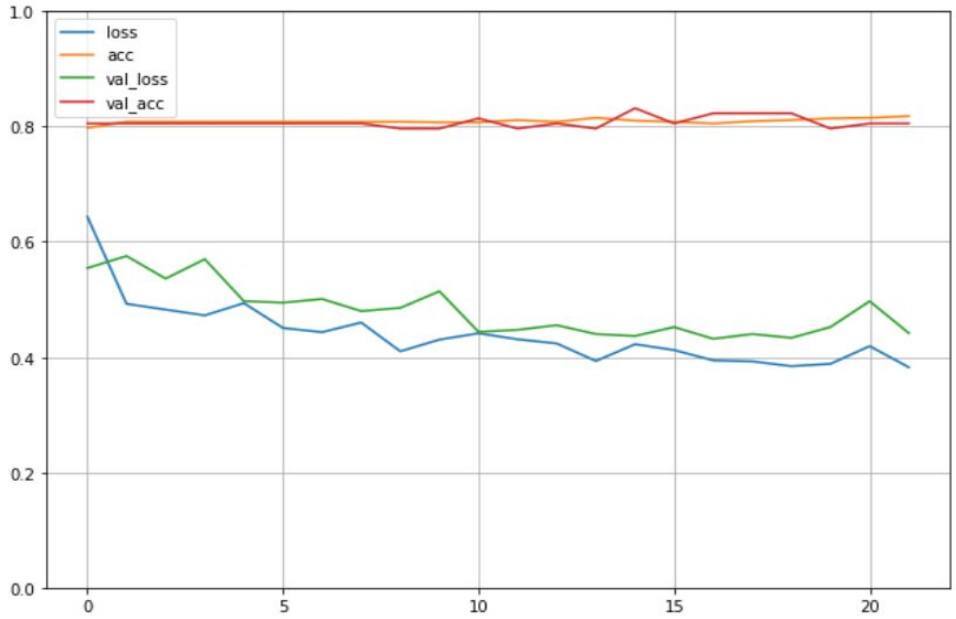
- Use regularization = 0.001 and no dropout layers



- Use SGD optimizer



- The executed model (learning rate = 10^-4 using the adam optimizer and remove some dropout layers)




```
<tensorflow.python.keras.callbacks.History at 0x7ff76a0f4b38>
```

```
model_temp1 =processing.load_model(processing.vgg,processing.axial,processing.abnormal,processing.extractor)
list1 = processing.test_extractor(model_temp1,processing.axial,processing.abnormal)
```

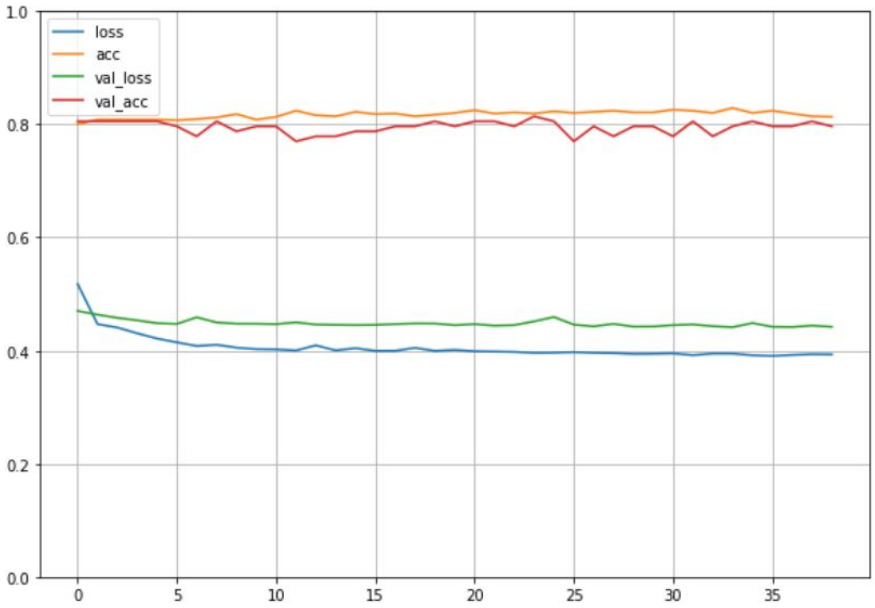
```
4/4 [==============================] - 0s 71ms/step - loss: 0.4750 - acc: 0.8167
```

The classifier :

After loading the extractor , we need to connect it to a new classifier (not trained before) to train this new model on many slices for a specific anomaly according to each series and getting the probability of this anomaly according to a specific series.

Train Classifier trials:

- Using 4096 neurons in each dense layer

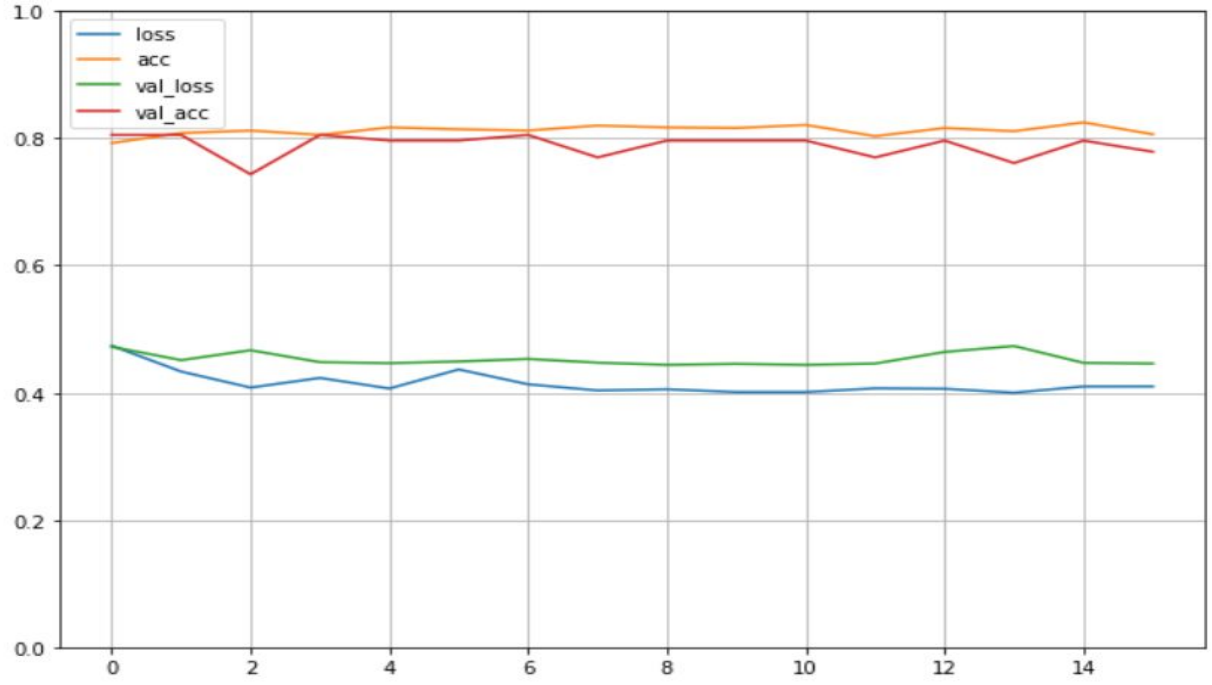



```
<tensorflow.python.keras.callbacks.History at 0x7f78a8c7eef0>
```

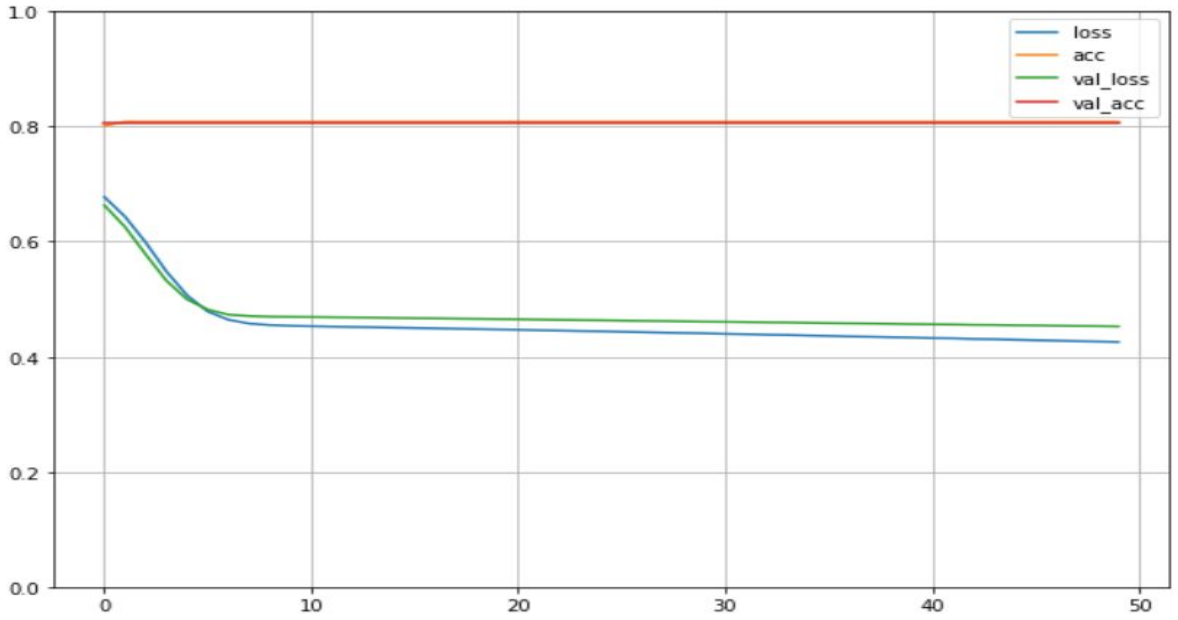
```
processing.test_classifier(model_axial_abnormal, model_classifier ,processing.axial , processing.abnormal)
```

```
4/4 [==============================] - 0s 4ms/step - loss: 0.4962 - acc: 0.7917
[0.4961872696876526, 0.7916666865348816]
```

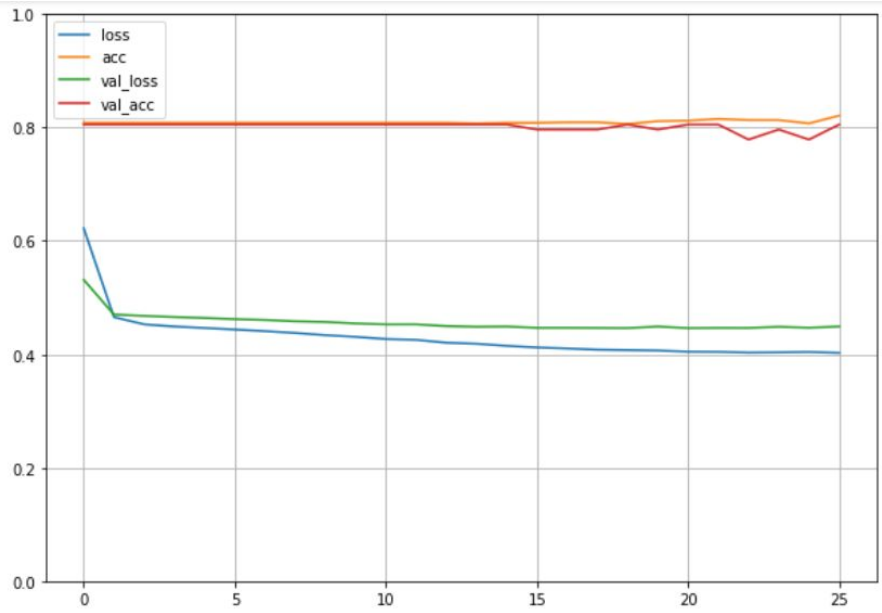
- increasing learning rate from 0.0001 to 0.01 but it didn't affect the performance



- Using 128 neurons on first dense layer and then using 64 neurons but it increases the loss a little bit



- Executed model (1024 neurons in first dense layer then 512 neurons in the second dense layer with learning rate = 10^-4)




```
<tensorflow.python.keras.callbacks.History at 0x7fa44949d588>
```

```
processing.test_classifier(model_axial_abnormal, model_classifier ,processing.axial , processing.abnormal)
```

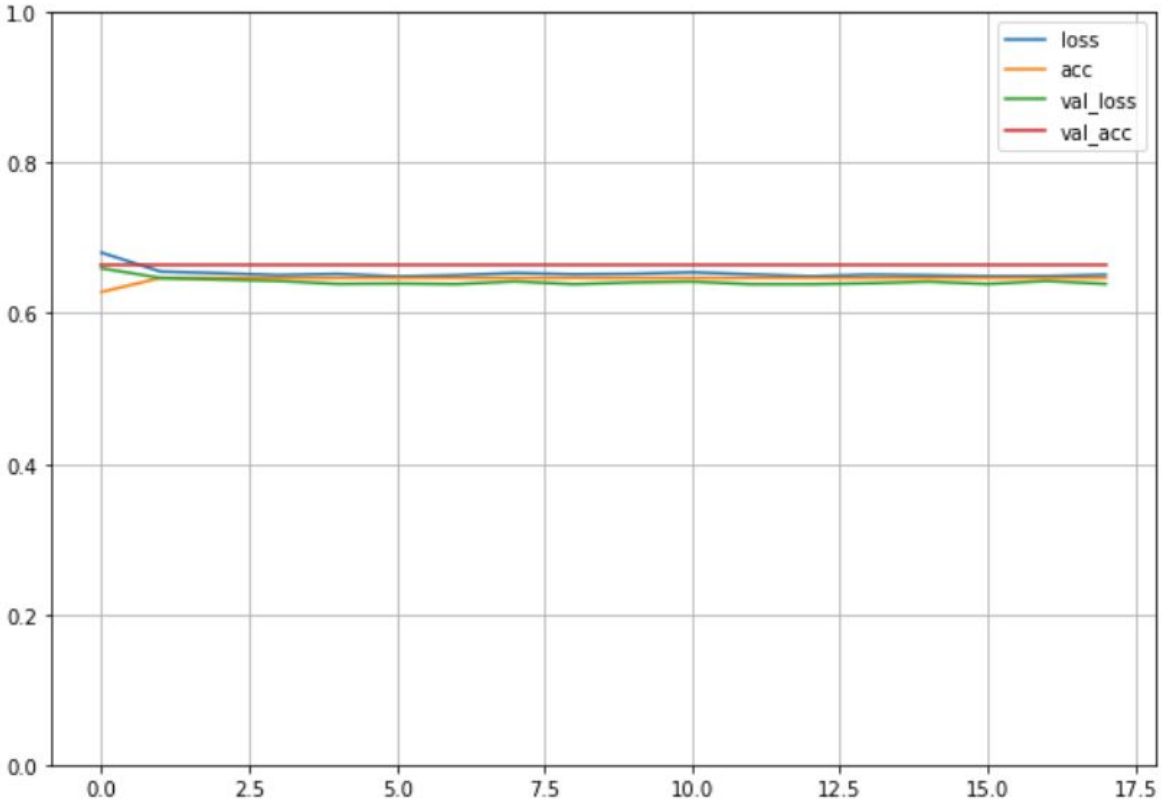
```
4/4 [==============================] - 0s 2ms/step - loss: 0.4918 - acc: 0.8000
[0.4918324649333954, 0.800000011920929]
```

The Regressor :

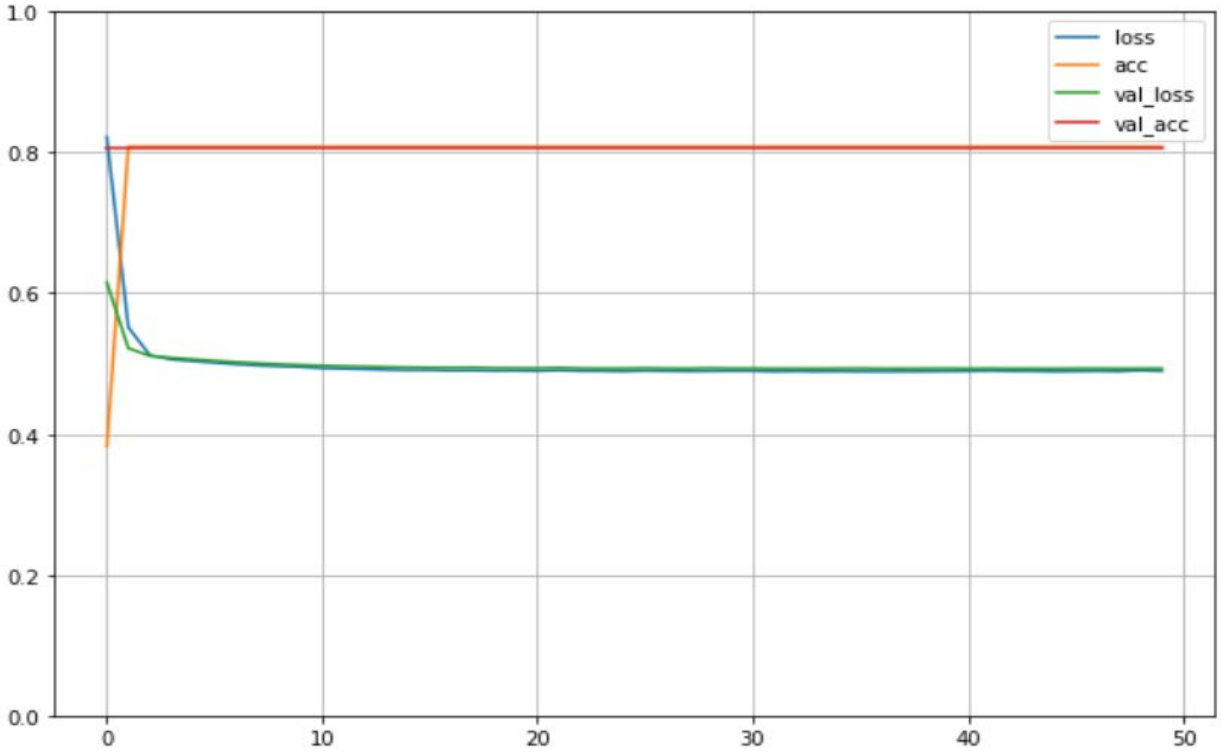
It produces the probability of each anomaly by taking the three inputs (outputs of the classifier which is the probability of the anomaly according to a specific series) for a specific anomaly from each series (axial , sagittal and coronal)

Train Regressor trials:

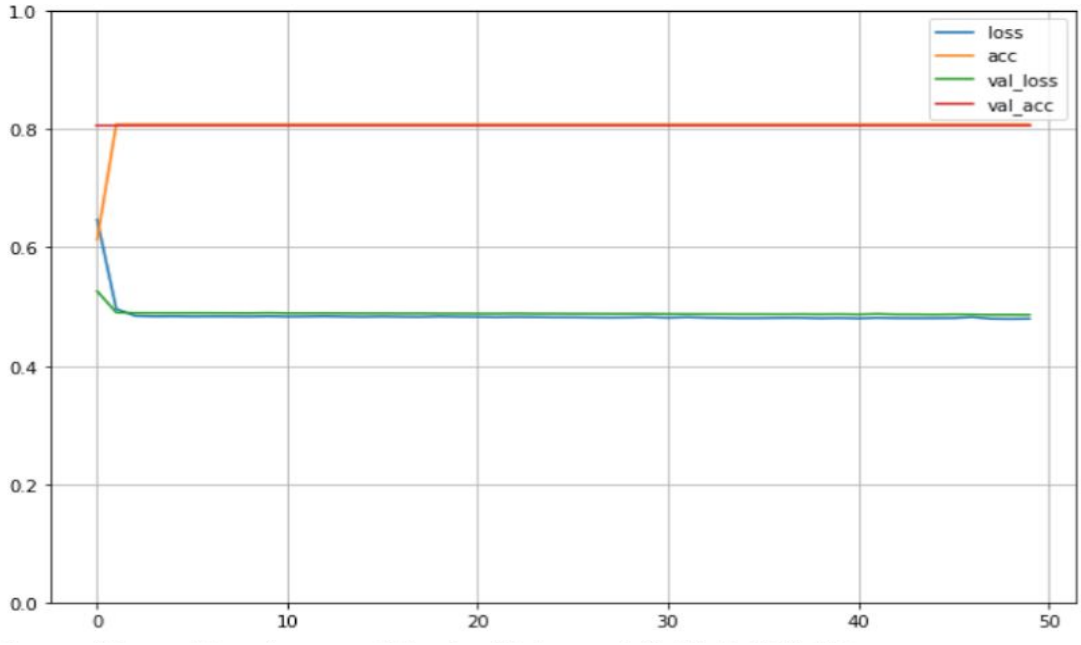
- With low learning rate = 10^-4 causes high loss



- Use regularization term = 0.01

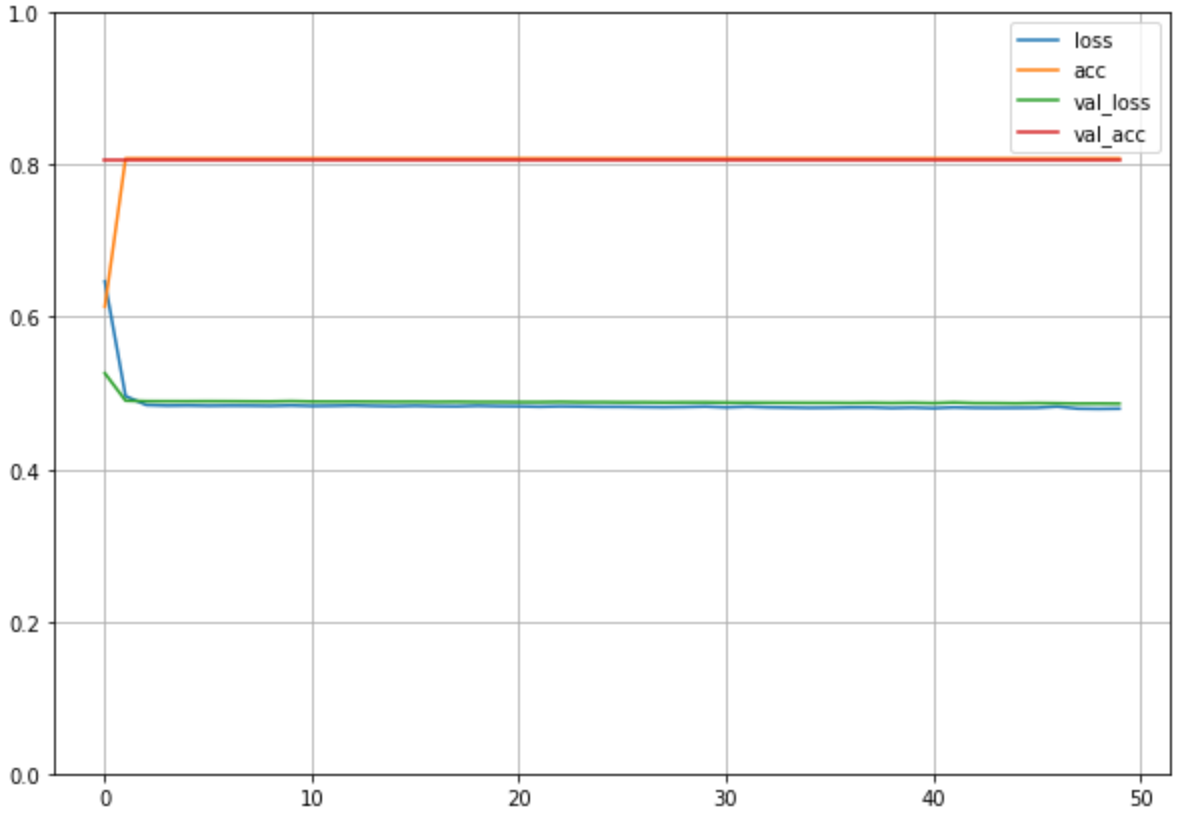


- Executed model (learning rate = 0.01 and without regularization)



Training Regressors:

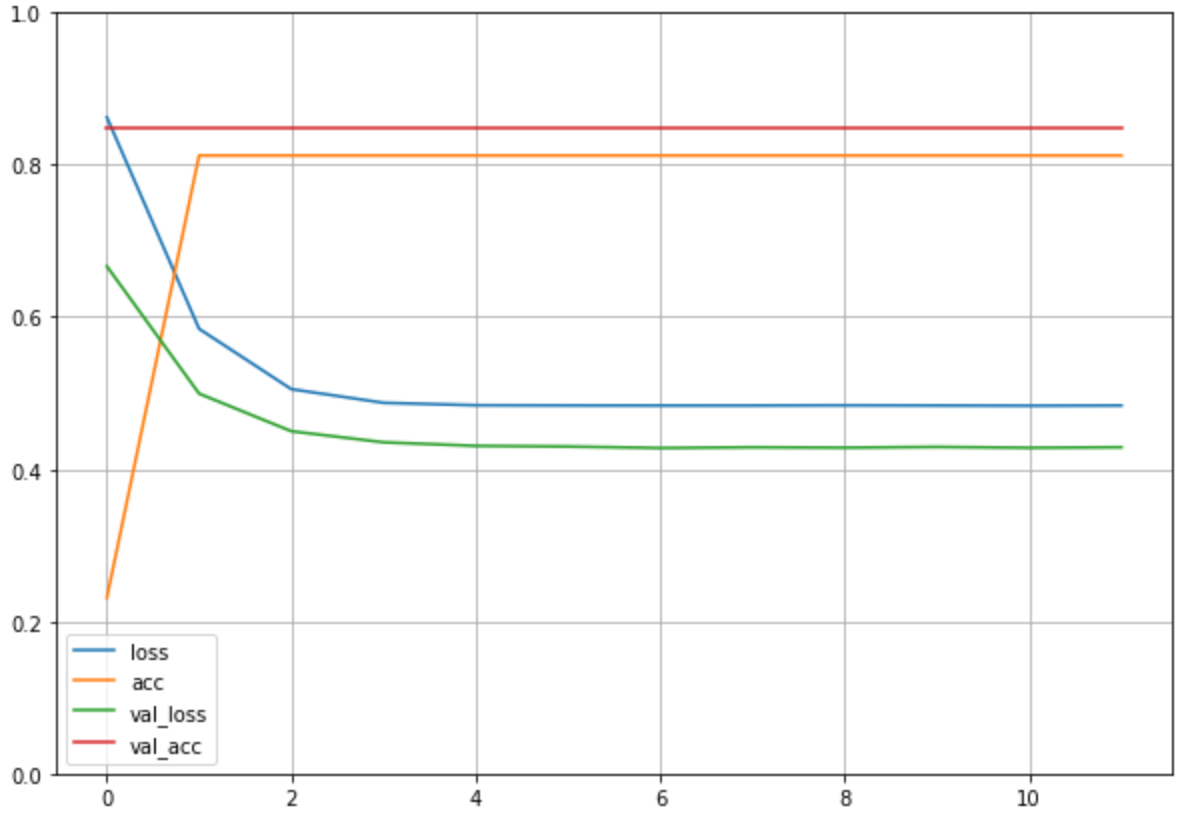
Abnormal:




```
processing.test_regressor(loaded_model_axial_abnormal,loaded_model_saggital_abnormal,loaded_model_coronal_
abnormal,processing.vgg,processing.abnormal)
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.5356 - acc: 0.7917
[0.5355910658836365, 0.7916666865348816]
```

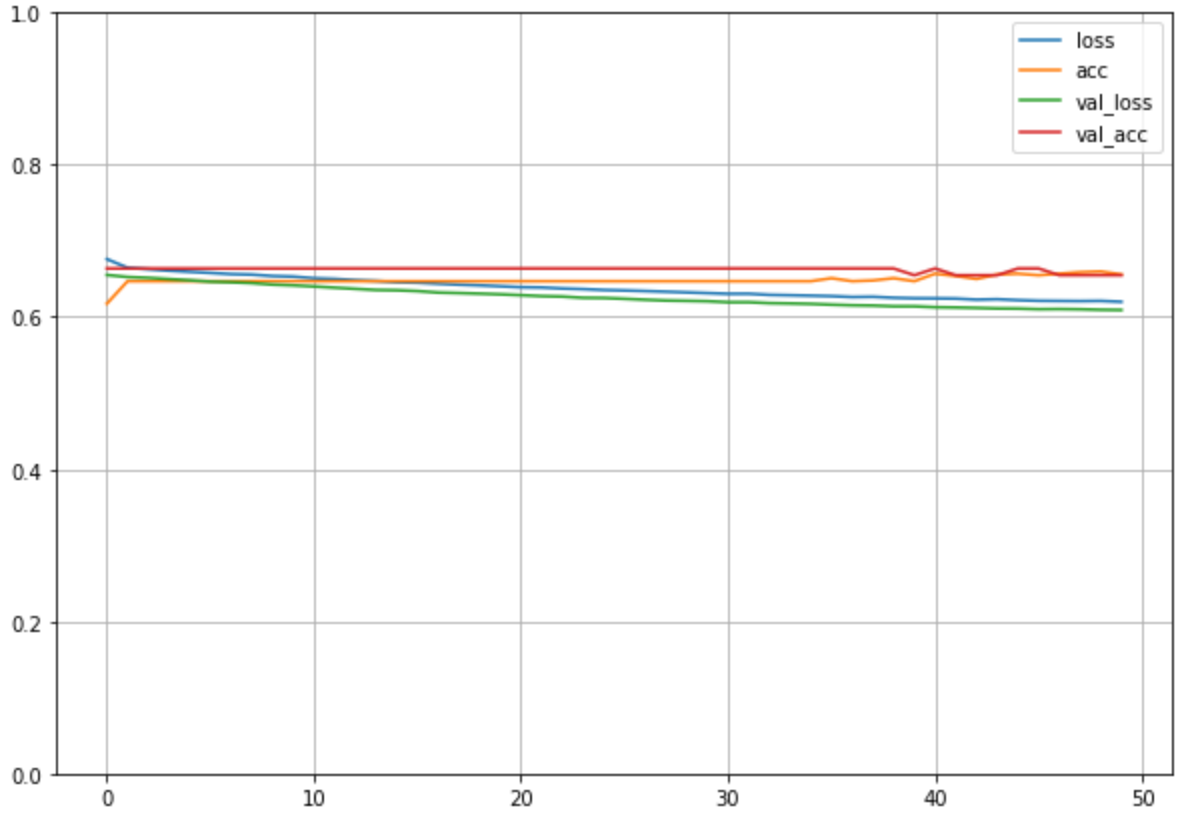
ACL:




```
processing.test_regressor(loaded_model_axial_acl,loaded_model_saggital_acl,loaded_model_coronal_acl,proces
sing.vgg,processing.acl)
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.6903 - acc: 0.5500
[0.6902922987937927, 0.550000011920929]
```

Meniscal:




```
processing.test_regressor(loaded_model_axial_meniscal,loaded_model_saggital_meniscal,loaded_model_coronal_
meniscal,processing.vgg,processing.meniscal)
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.6982 - acc: 0.5667
[0.6982162594795227, 0.5666666626930237]
```

Summary of VGG results:

	Accuracy	Loss
Abnormal	0.7917	0.5356
ACL	0.5500	0.6903
Meniscal	0.5667	0.6982

Transfer Learning :

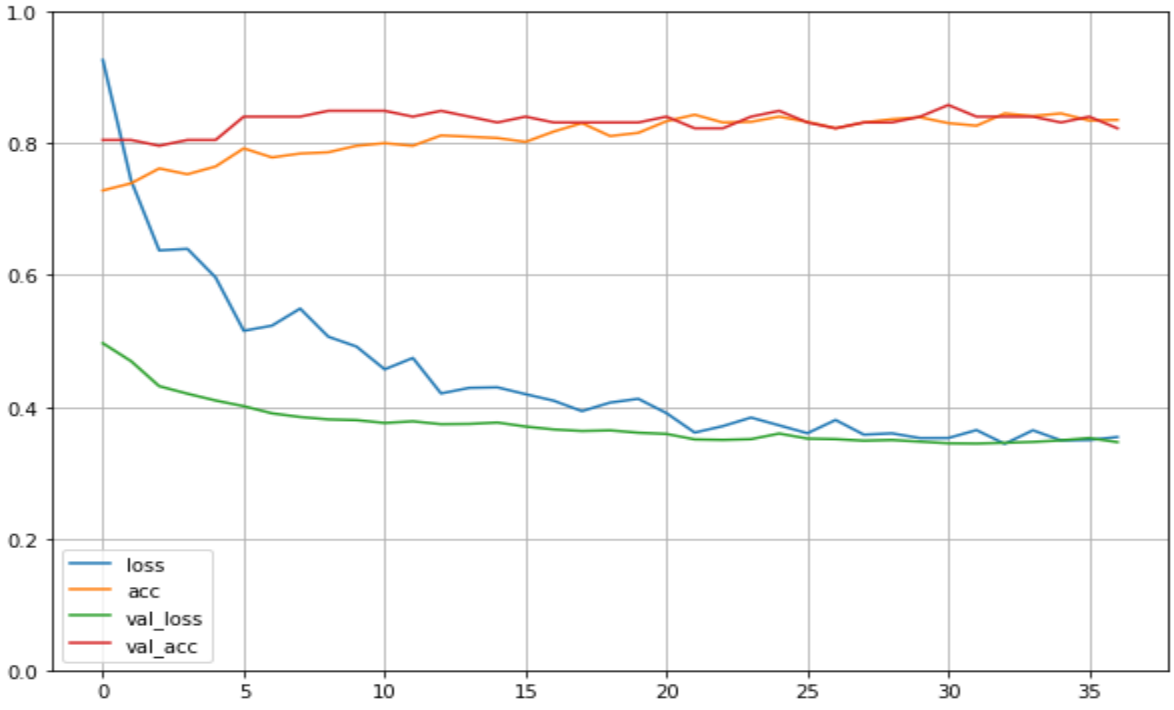
RESNet 50 :

I trained the extractors of the model using the RESnet model implemented in keras and trained it by using ImageNet dataset which contains more than 14 million different images same as the approach of the MRNet paper (but they used Alexnet model) and then we trained the classifier and regressor by our dataset (MRI images).

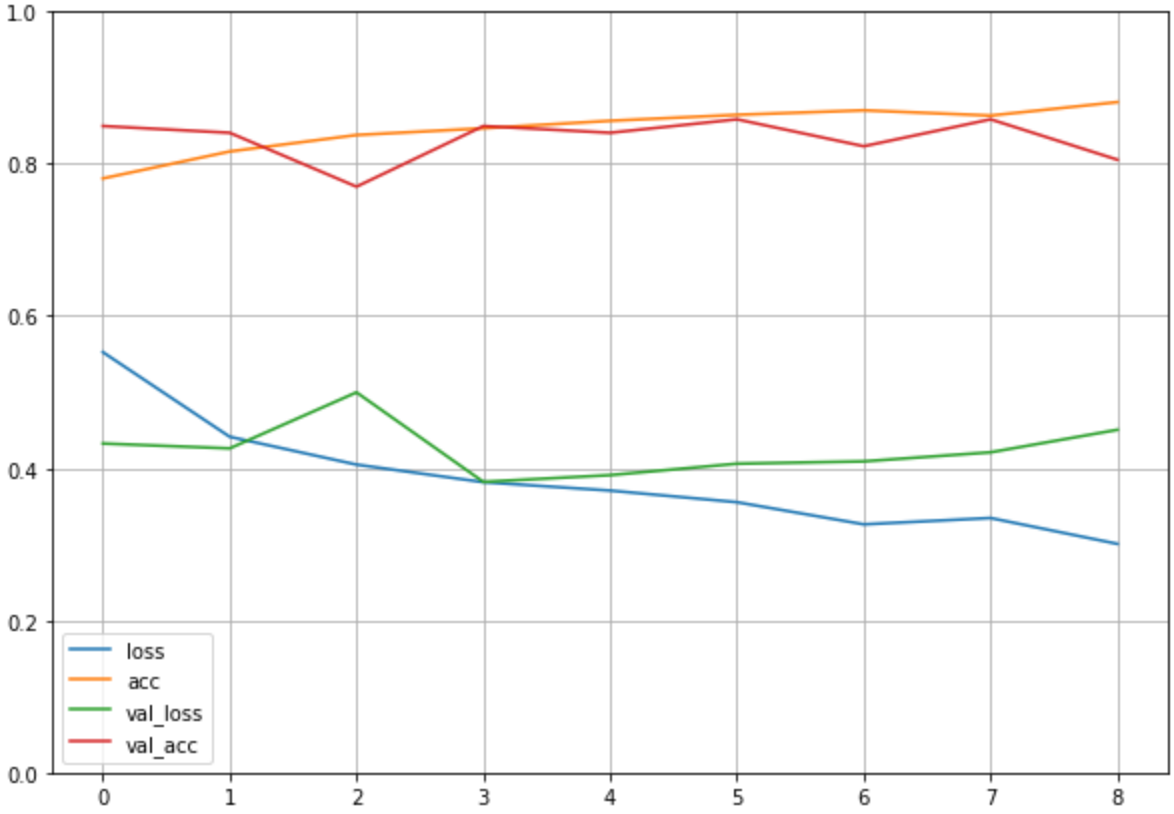
We used the transfer learning for our three models (VGG , RESNet and Inception v3) as they produce nearly the same results and we couldn't identify precisely which model is the best.

Transfer Classifier trials :

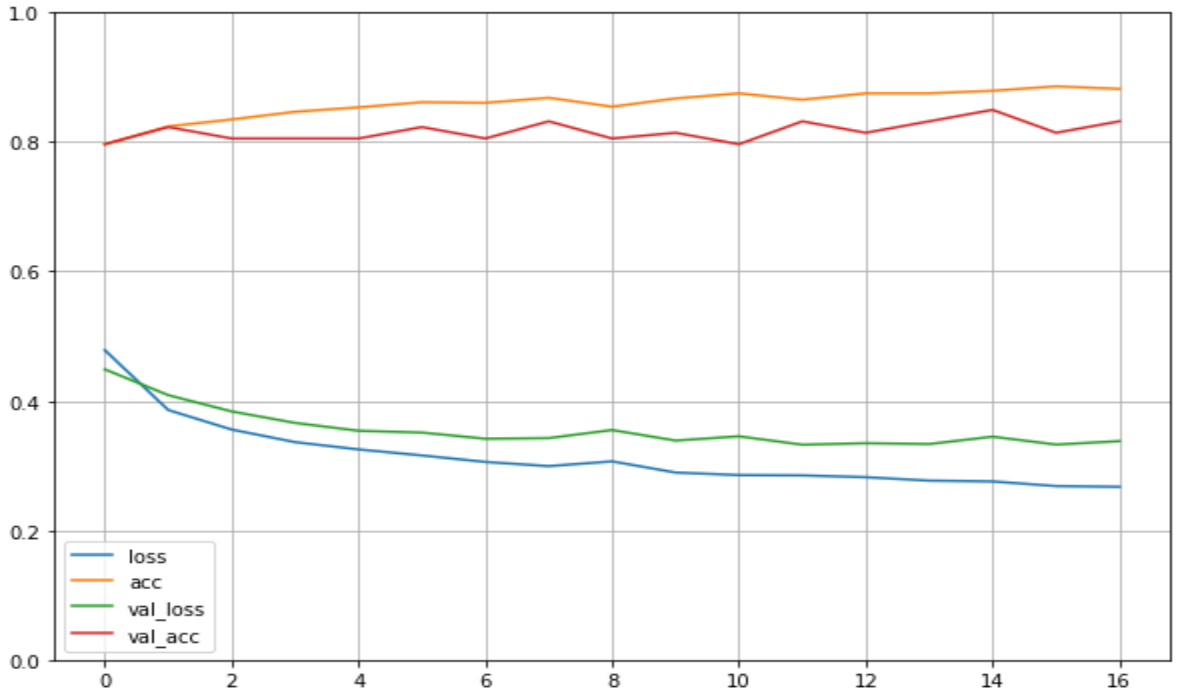
- (512 neurons in the first layer and 256 in the second one) and add dropout layers and learning rate = 10^-5 . the results are good but i try to reduce the loss



- By setting the learning rate to 10^-4 (the test is done for axial - ACL classifier)



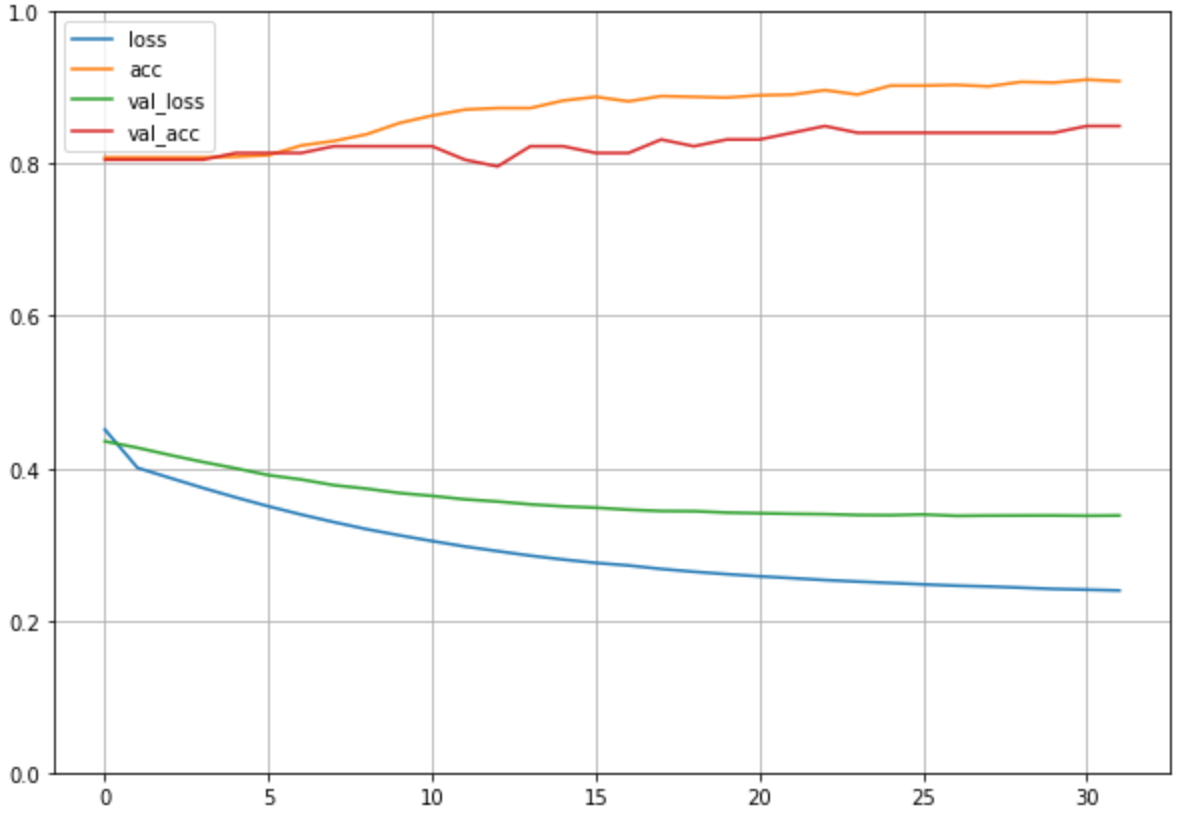
- The final executed model for classifier by using (512 neurons in first layer and 256 in the second one) and remove all dropout layers and learning rate = 10^-5



Train Regressor:

The executed model for regressor is one dense layer with one neuron with activation sigmoid function (output which detects the probability of the anomaly)

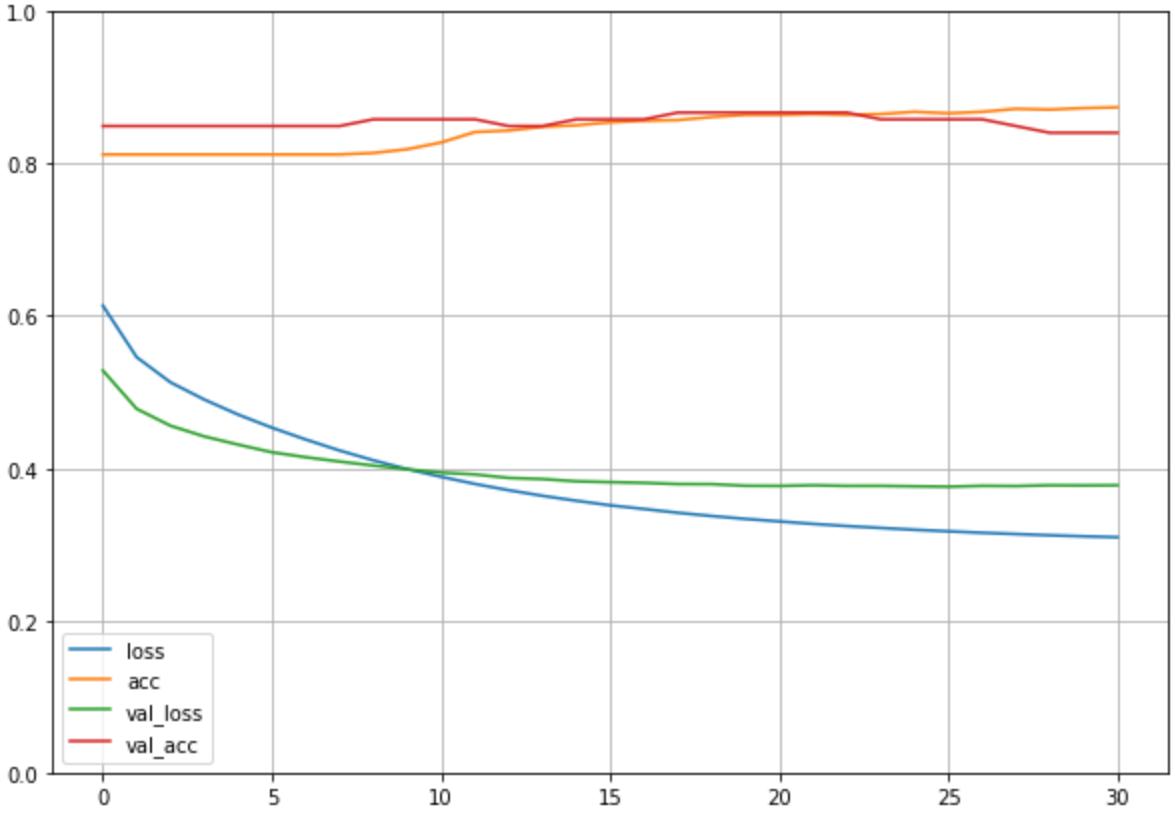
Abnormal:




```
processing.test_regressor(extractor_res,extractor_res,extractor_res,processing.trans_resnet,processing.abnormal)
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.3940 - acc: 0.8333
[0.39404281973838806, 0.8333333134651184]
```

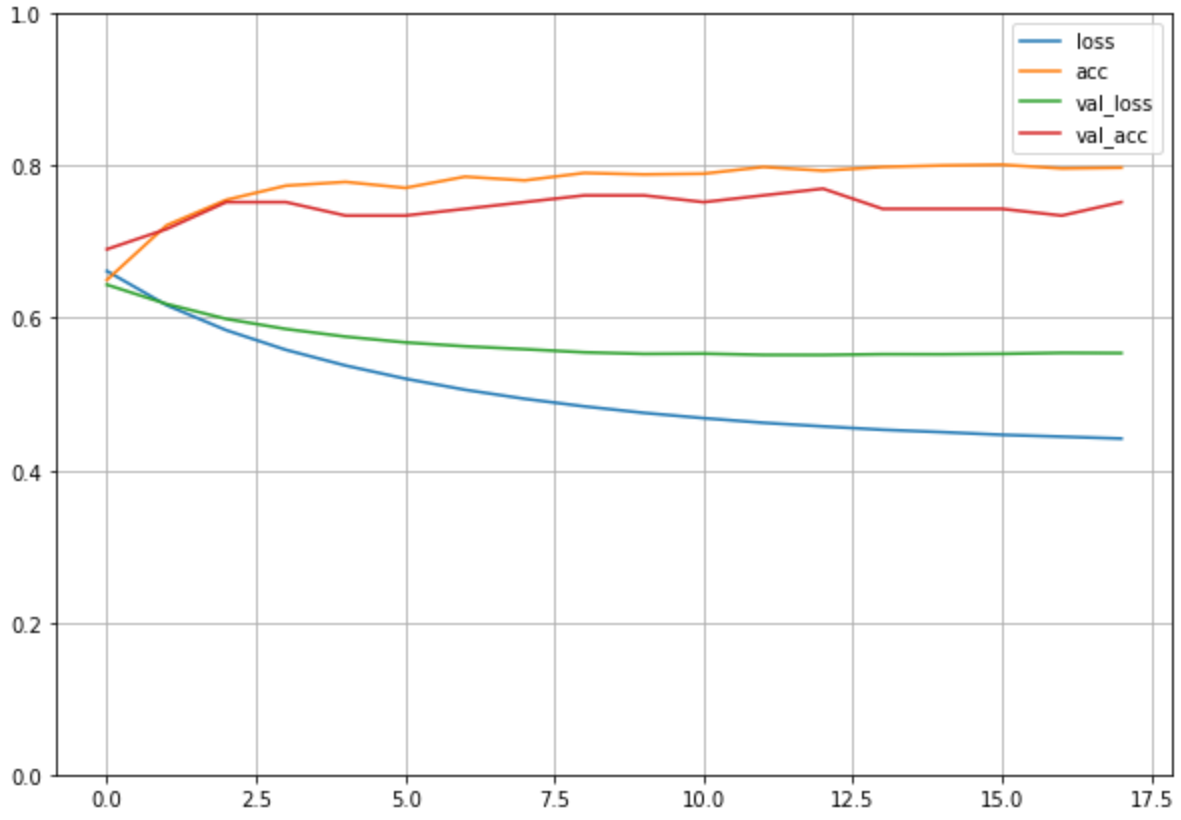
ACL:




```
processing.test_regressor(extractor_res,extractor_res,extractor_res,processing.trans_resnet,processing.acl)
```

```
4/4 [==============================] - 0s 3ms/step - loss: 0.6274 - acc: 0.6583
[0.6274384260177612, 0.6583333611488342]
```

Meniscal:




```
processing.test_regressor(extractor_res,extractor_res,extractor_res,processing.trans_resnet,processing.men
iscal)
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.5947 - acc: 0.7083
[0.5947383046150208, 0.7083333134651184]
```

Summary of RESNet results by transfer learning :

	Accuracy	Loss
Abnormal	0.8333	0.3940
ACL	0.6583	0.6274
Meniscal	0.7083	0.5947

The resnet model achieved the best results among all the trained models for the three different anomalies

Contribution (First paper):

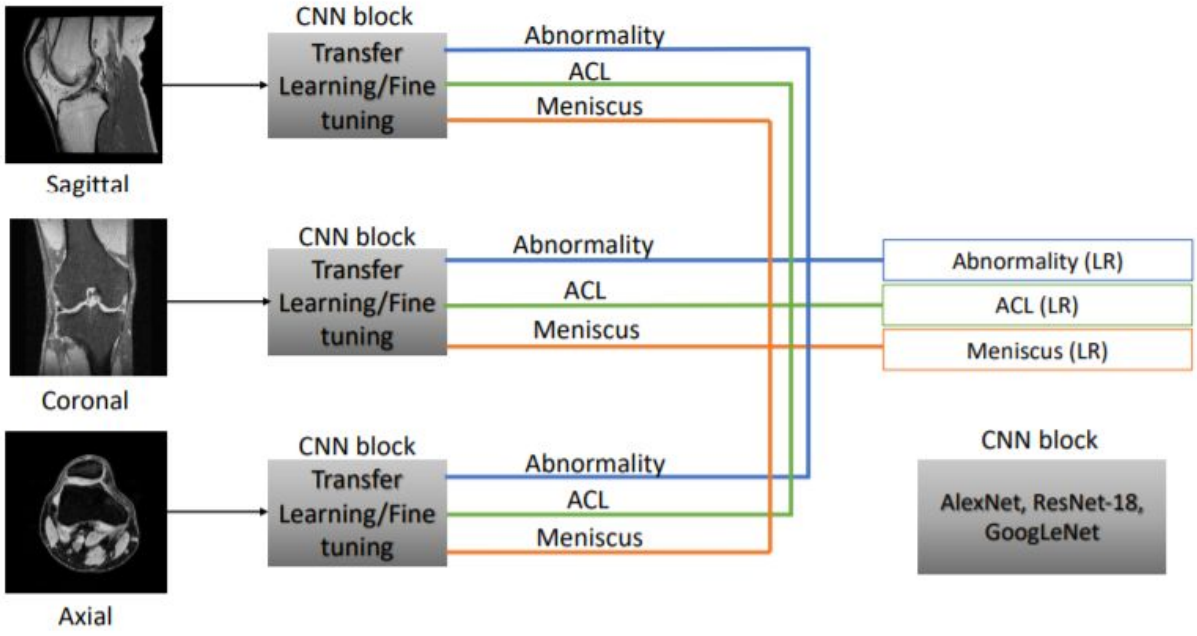
I read the first paper suggested in the contribution section of the project pdf which is “Deep Learning for Musculoskeletal Image Analysis”.

The paper used the same model as the MRNet paper by transfer learning (training extractors on Imagenet) but in addition to AlexNet used as a feature extractor , they also used GoogleNet and ResNet-18 convolutional neural networks.

They also used the same number of MRI exams as the MRNet paper (1370 exams).

The statistics showed that the RESNet model gives the best results for all statistical methods used(area under curve , sensitivity and accuracy) except the specificity.

I nearly used the same approach by using the RESNet model for training the extractors by transfer learning.



Finally the paper suggested Two methods that can help in improving the performance of models which are :

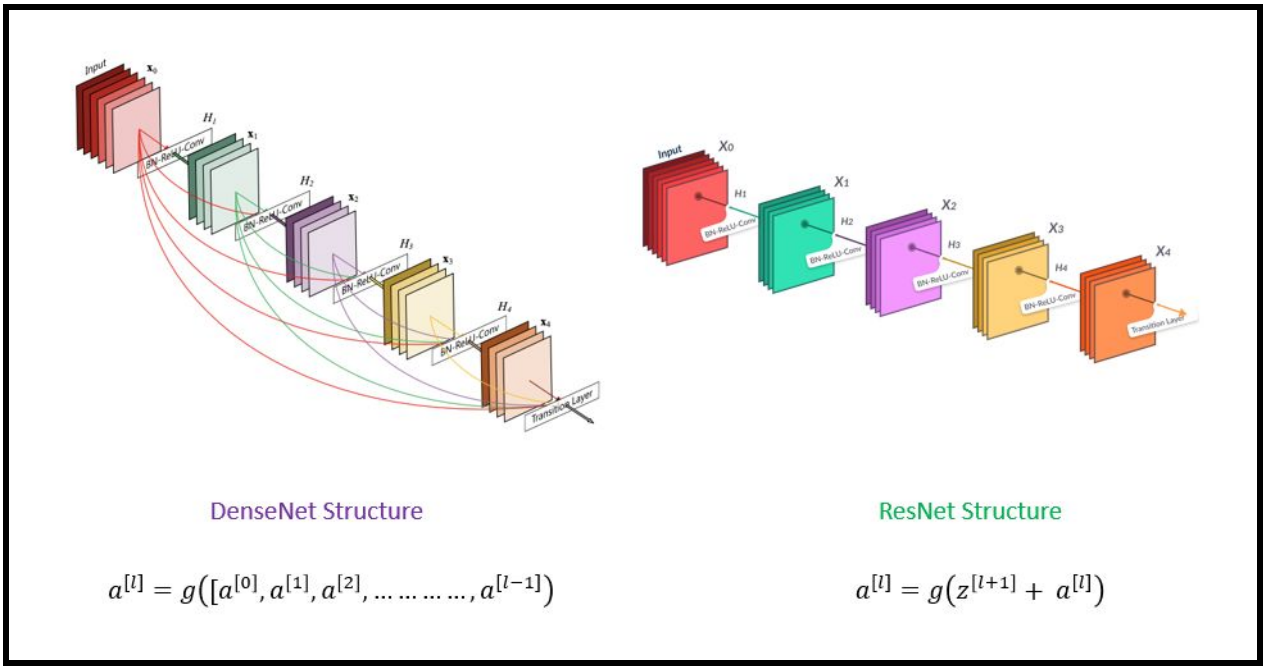
- Using advanced architecture such as DenseNet or CapsNet as feature extractors
- Using a large amount of imaging data.

The second solution is an extreme challenge so I tried to use the first approach by using transfer learning for training the feature extractors by using the DenseNet model implemented in keras.

DenseNet Model:

It is considered as development for the RESNet model but instead of adding activation of a layer to a further one , the Functions are concatenated and each layer is connected to all subsequent layers so the total connections are L (L+1) / 2 where L is the number of layers in the network.

It requires fewer parameters than traditional convolutional networks because there is no need to relearn redundant feature maps.its architecture helps in this advantage instead of passing a state from one layer to the next one only as in traditional convolutional networks and it also helps in improved flow of information and gradient in the network

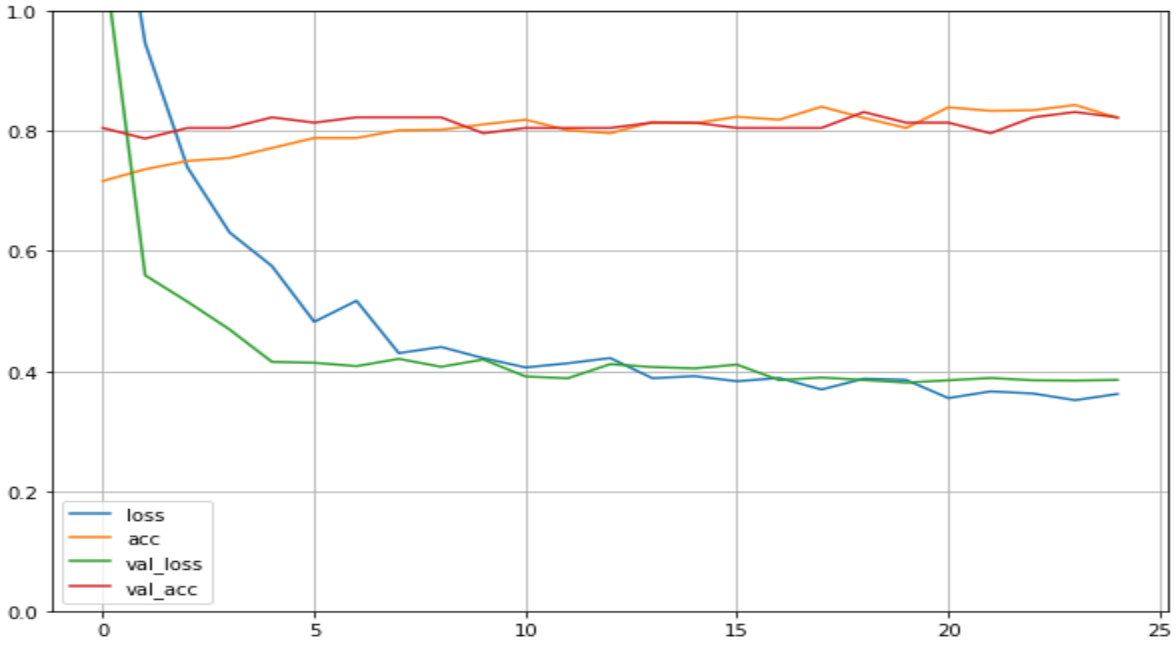


DenseNet blocks:

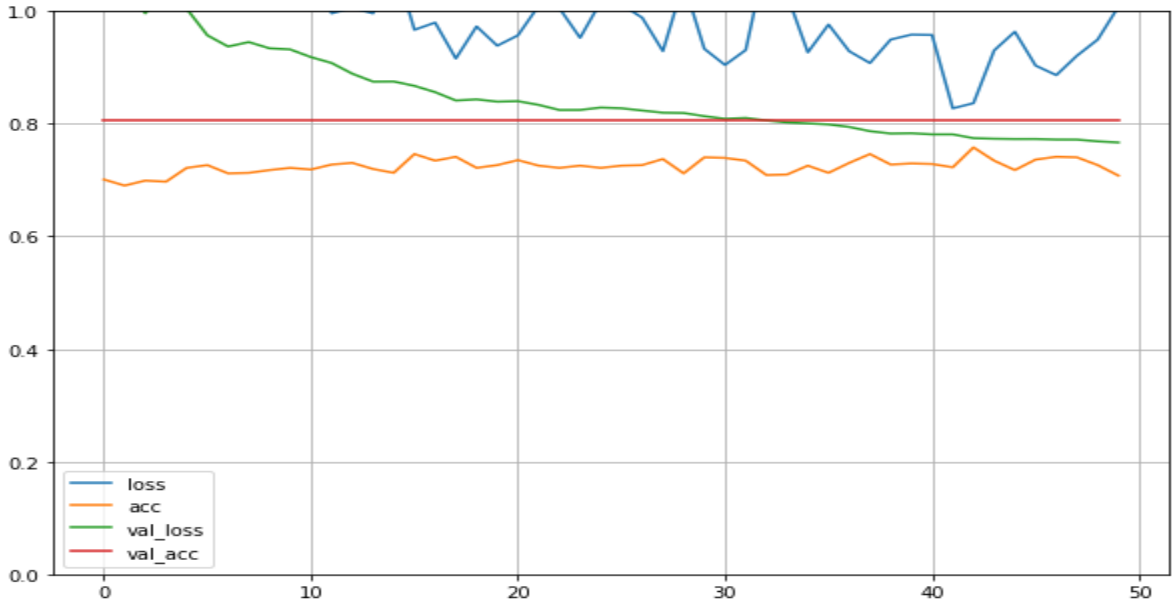
Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 36
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 24
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Train Classifier trials:

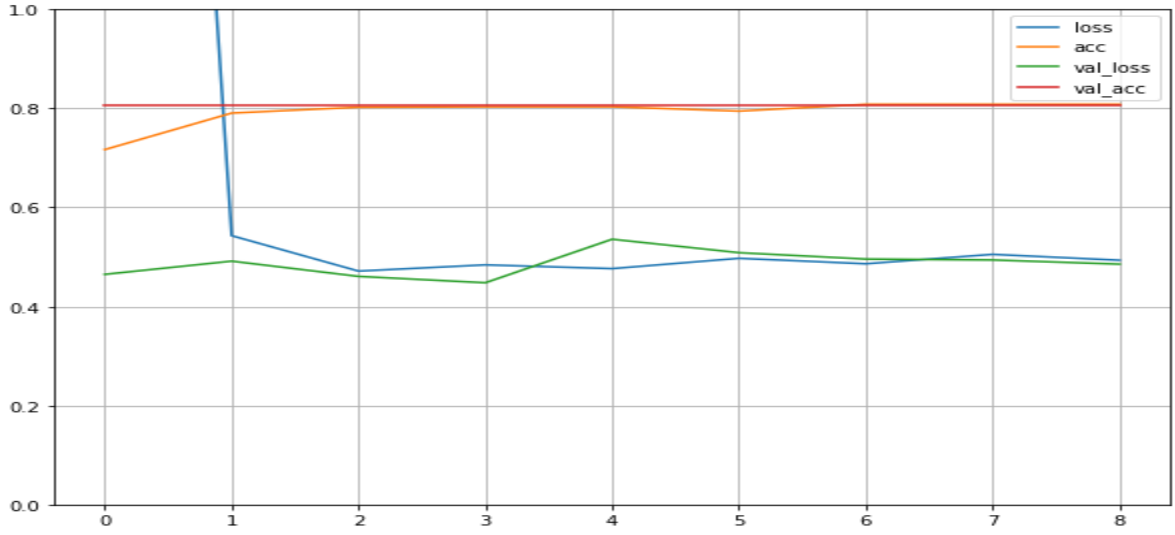
- Use (512 neurons in first layer and 256 in the second one) and add dropout layers to avoid overfitting



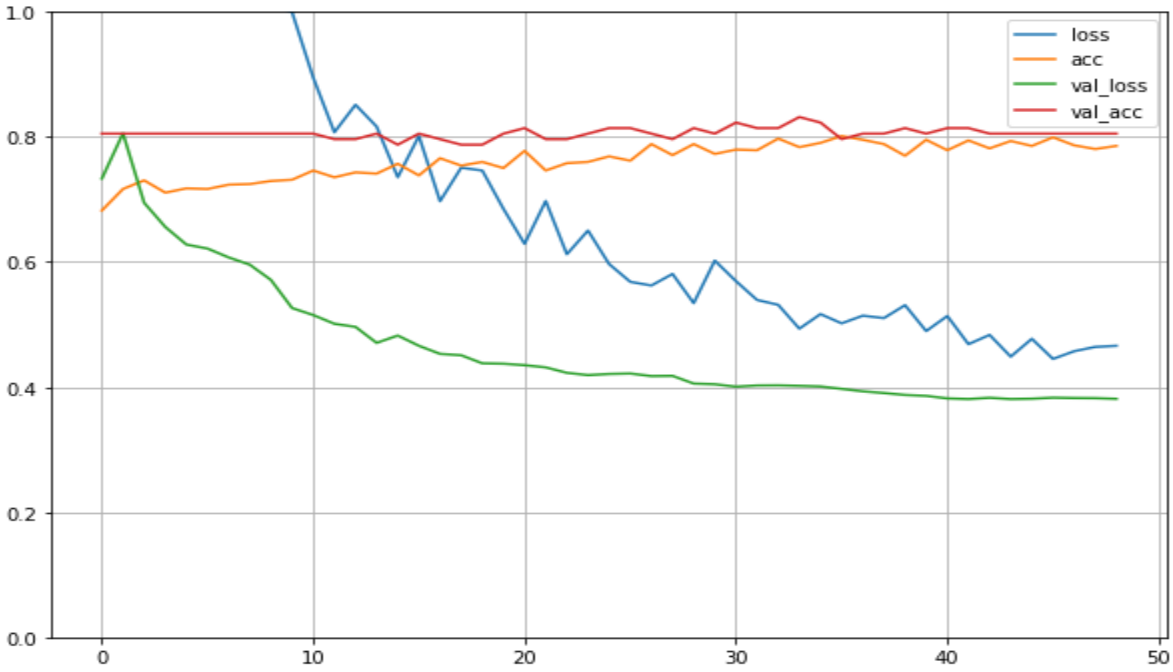
- Using learning rate = 10^-4 and decay = 0.1 but it makes the loss very high due to decreasing the learning rate but the loss decays slowly



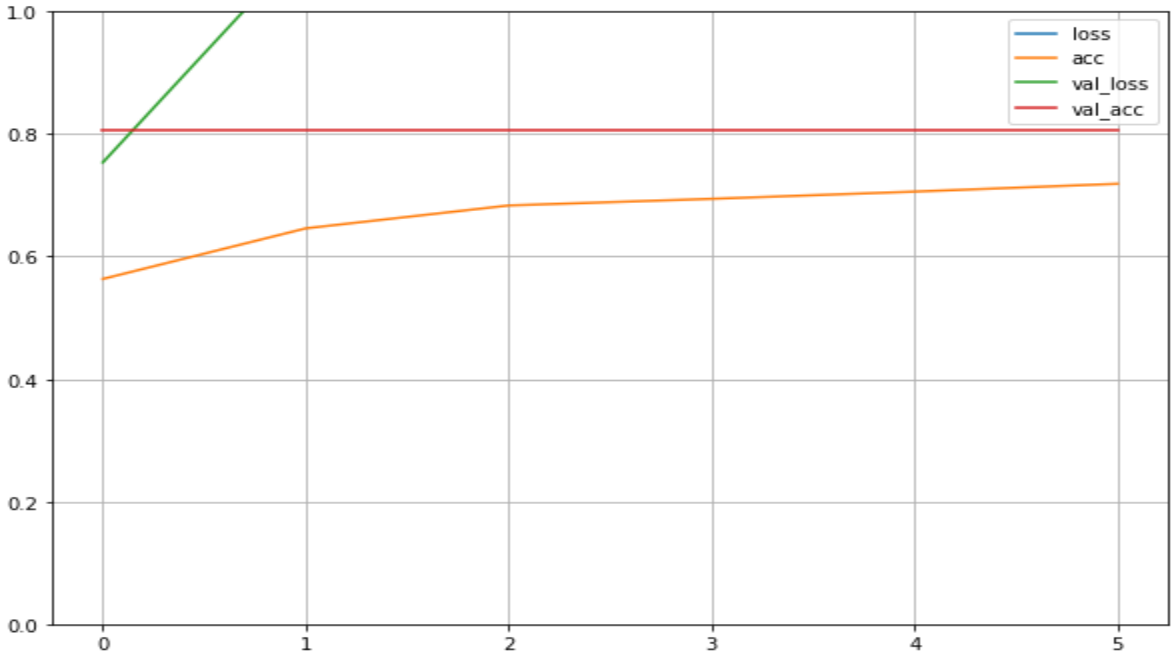
- Using learning rate = 0.01 with the same neurons and dropout layers . here I try to increase the learning rate but the loss decreases quickly at the first and then became nearly constant



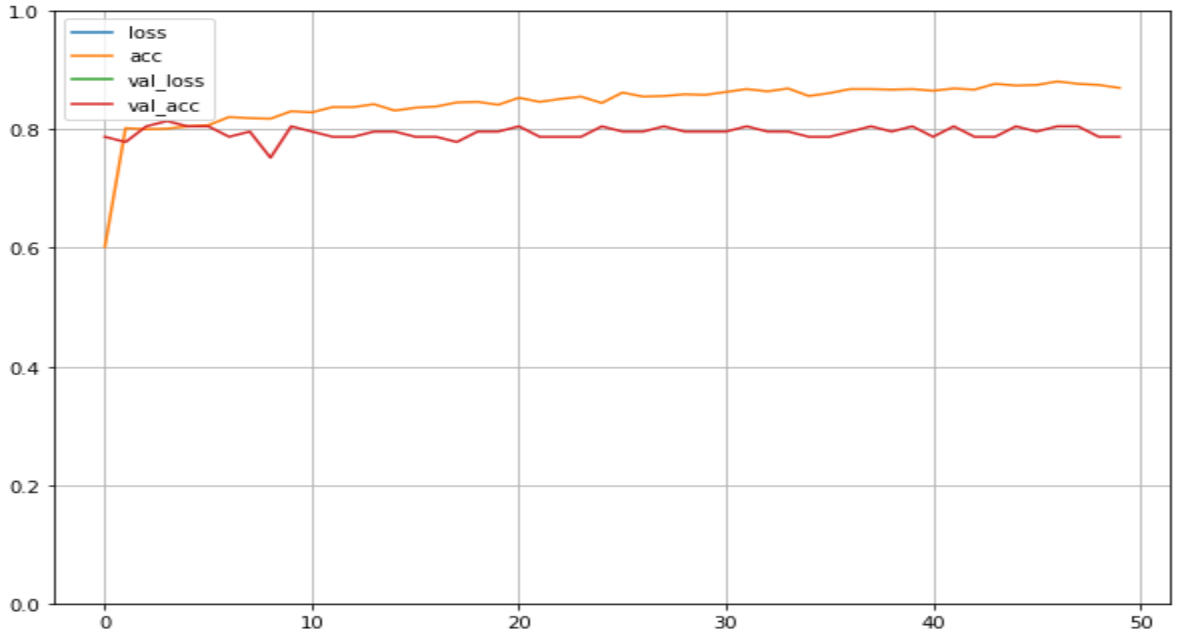
- Using learning rate = 10^-5 with the same neurons and dropout layers . I decreased the learning rate to make curve of loss more smoothly but the loss was a little bit high



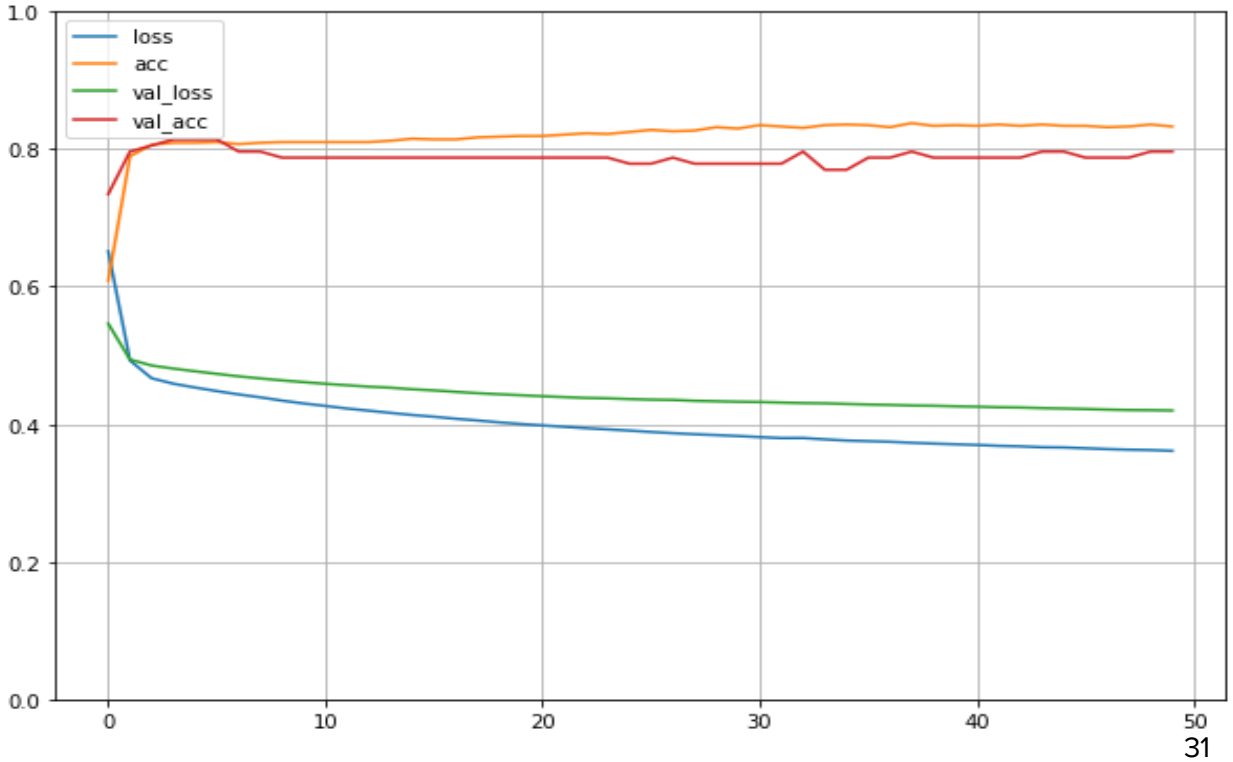
- Use (512 neurons in first layer and 128 in the second one) and add dropout layers with learning rate = 10^-5 but loss became very high



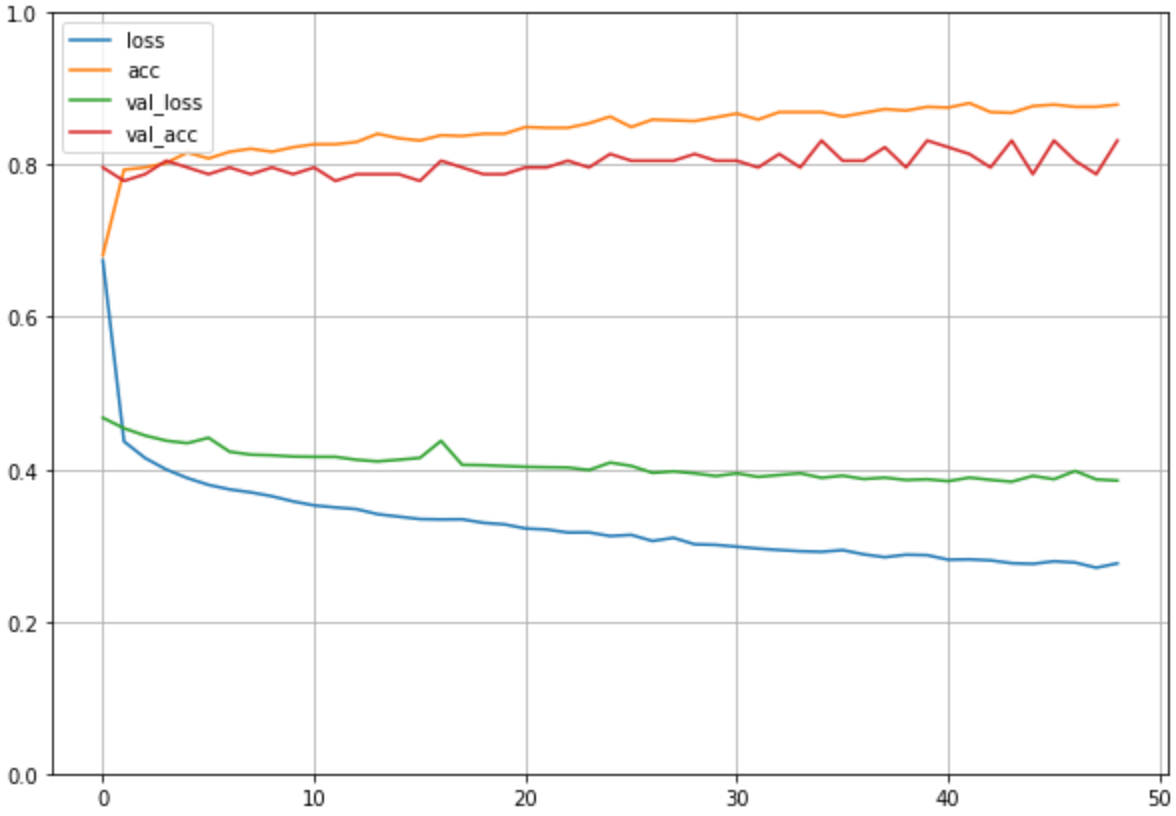
- Using regularization but the loss is still very high



- With learning rate = 10^-6 and removing regularization and dropout layers. The loss is still high so i try to decrease it more in the next tries



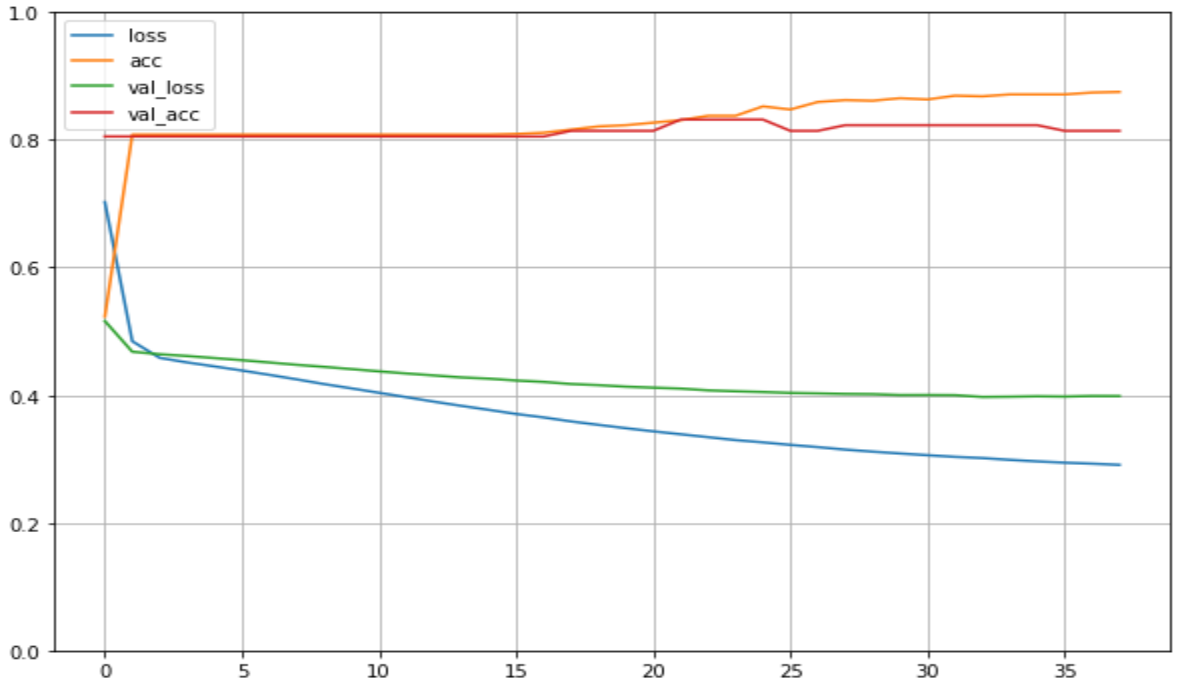
- The final executed model for classifier by using (512 neurons in first layer and 256 in the second one) and remove all dropout layers and learning rate = 10^-5



Train Regressor:

The executed model for regressor is one dense layer with one neuron with activation sigmoid function (output which detects the probability of the anomaly)

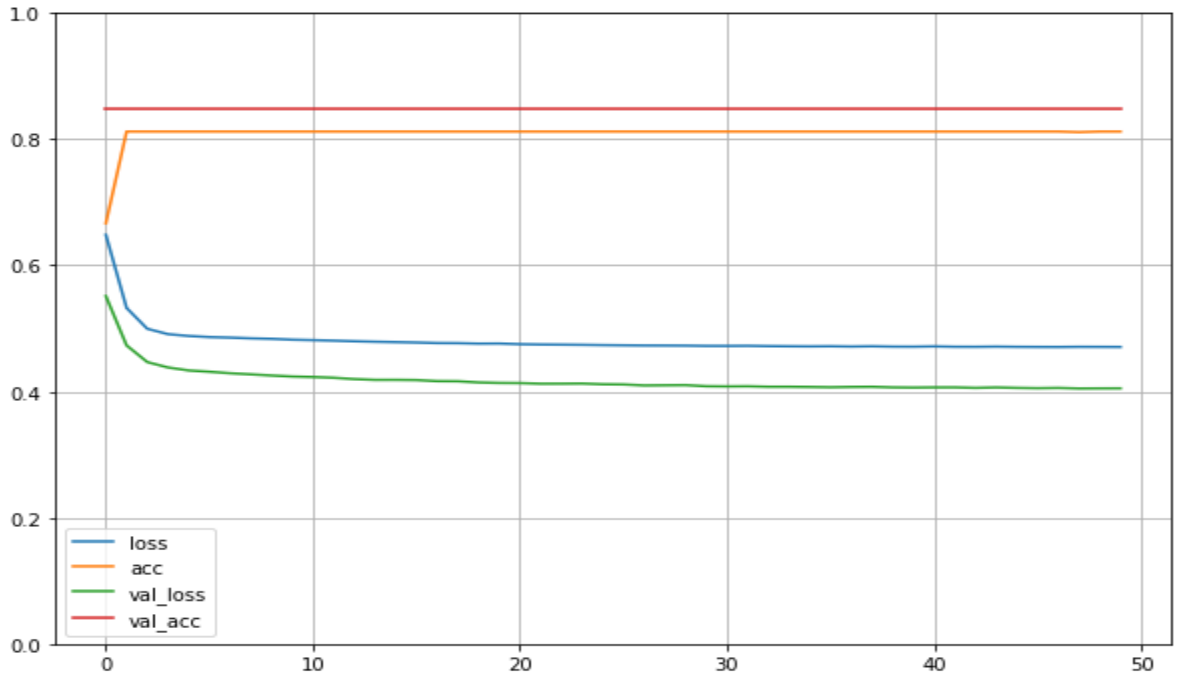
Abnormal:




```
processing.test_regressor(extractor_dense,extractor_dense,extractor_dense,processing.densenet,processing.abnormal)
```

```
4/4 [==============================] - 0s 4ms/step - loss: 0.4418 - acc: 0.8167
[0.4417559802532196, 0.8166666626930237]
```

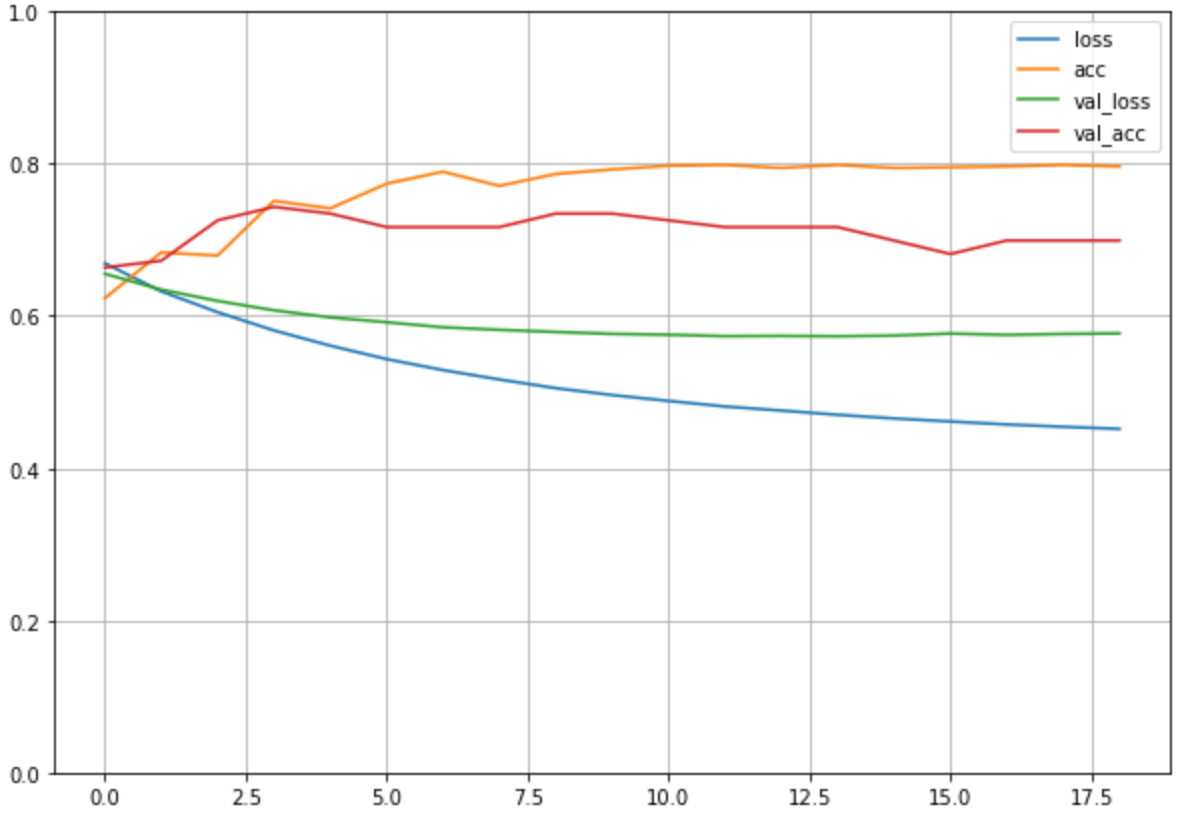
ACL:




```
processing.test_regressor(extractor_dense,extractor_dense,extractor_dense,processing.densenet,processing.acl)
```

```
4/4 [==============================] - 0s 4ms/step - loss: 0.6972 - acc: 0.5500
[0.6971763372421265, 0.550000011920929]
```

Meniscal:




```
processing.test_regressor(extractor_dense,extractor_dense,extractor_dense,processing.densenet,processing.meniscal)
```

```
4/4 [==============================] - 0s 4ms/step - loss: 0.6453 - acc: 0.6417
[0.6452761888504028, 0.6416666507720947]
```

Summary of DenseNet results by transfer learning :

	Accuracy	Loss
Abnormal	0.8167	0.4418
ACL	0.5500	0.6972
Meniscal	0.6417	0.6453