POLITECNICO DI TORINO

Electronic Engineering

BIOINFORMATICS

Author:
Dimroci Enea

Prof:
Elisa Ficarra

October 2020

Contents

0.1 ASSIGNMENT DESCRIPTION 2
0.2 GENOMICS 2
0.2.1 Gene Expression 2
0.2.2 Data Preparation 3
0.2.2.1 Download Data 3
0.2.3 Data Preprocessing 4
0.2.3.1 Balance Healthy and Tumoral Data 4
0.2.3.2 Standard Preprocessing 5
0.2.3.3 Principal Component Analysis 5
0.2.4 Model Definition 7
0.2.5 Workflow 9
0.2.6 Results 9
0.3 IMAGING 11
0.3.1 Whole Slide Image 11
0.3.2 Data Preparation 12
0.3.3 Data Preprocessing 13
0.3.4 Benchmark Models 14
0.3.5 Workflow 15
0.3.6 Results 16
0.3.7 My Model 20
0.3.8 Results 25
0.4 CONSENSUS CLASSIFIER 26
0.5 References 29

Cross-over bioimaging and genomics – Consensus classifier

0.1 ASSIGNMENT DESCRIPTION

The aim of this project is to classify colorectal cancer with the integration of informations coming from Gene Expression analysis and from Whole Slide Image analysis. This integration is done building a consensus classifier that manage the results coming from the mRNA classification and the Whole Slide Image classification.

0.2 GENOMICS

0.2.1 Gene Expression

The *Gene Expression* is the process by which information is transfered from DNA to RNA (*Transcription*) and from RNA to protein (*Translation*). So it is the phenotypic manifestation of genes.

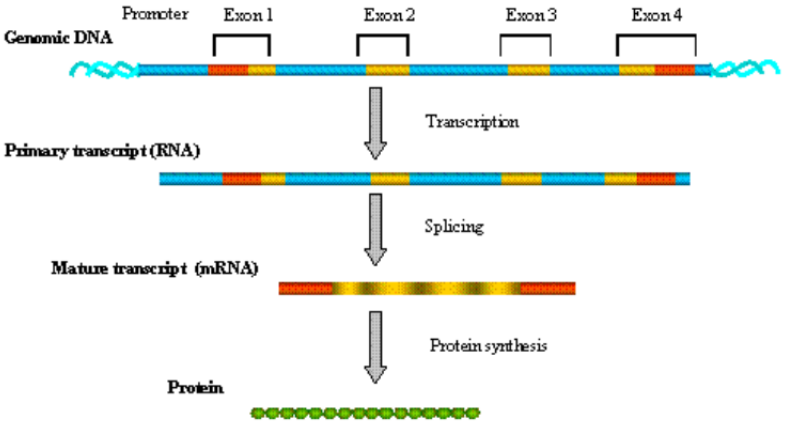

Figure 1: GENE EXPRESSION

This measurement can be done with the *Real-Time qRT-PCR* that is the major development of PCR technology.
At the end of this workflow, the data that we have are values that quantify the expression of different genes.

0.2.2 Data Preparation

0.2.2.1 Download Data

The data used for this project are downloaded from the GDC (Genomic Data Commons) Data Portal.
Different filters are applied in order to select the Gene Expression Quantification:

- Cancer Cases : Colorectal Cancer.
- Data Category : Transcriptom Profiling
- Data Type : Gene Expression Quantification
- Workflow Type : HTSeq FPKM

Data are effectively downloaded from the repository with the procedure explained by the GDC Data Transfer Tool Documentation.
For the cancer case 648 files are available instead of the healthy case which have only 41 files.
The data format is *.txt*, but each of them is also zipped in a *.gz* format.
In order to manage the data each zip file is opened, and the *.txt* is red, finally its content is stored in a row of a *Pandas DataFrame Structure*.
This is done for all the zipped files.
The *Features Vector* has 60483 elements, and the total number of samples is 682.
Once the features vector is determined is necessary to include the correct label for each sample, so a label column is added to the pandas dataframe.
So at this point the data structure with all the necessary information for the analysis is populated.
The first problem, that can leads to a wrong analysis, is that is necessary to balance these data because of the few healthy samples, only 42 over all the samples.
Otherwise all the performance results such as the accuracy, the precision and the recall are meaningless.

0.2.3 Data Preprocessing

0.2.3.1 Balance Healthy and Tumoral Data

First of all the train and test split has to be done, in this way some samples are left for the performance evaluation of the model.
So is possible to work on the training set in order to augment the healthy samples.
Ideally the same number of samples belonging to these two diffent classes has to be the same.
The idea is to create new samples copying the values of gene expressions on healthy samples.
But in this way, when we do the train and test split, the test set has always samples that belongs to the training set. Also in a K-Fold Crossvalidation.
An over-sampling strategy is needed because of the few samples on the healthy class.
The strategy adopted is the so called *SMOTE* oversampling. This technique is able to identify a geometrical region in which the samples are placed, and creates new samples inside this region:

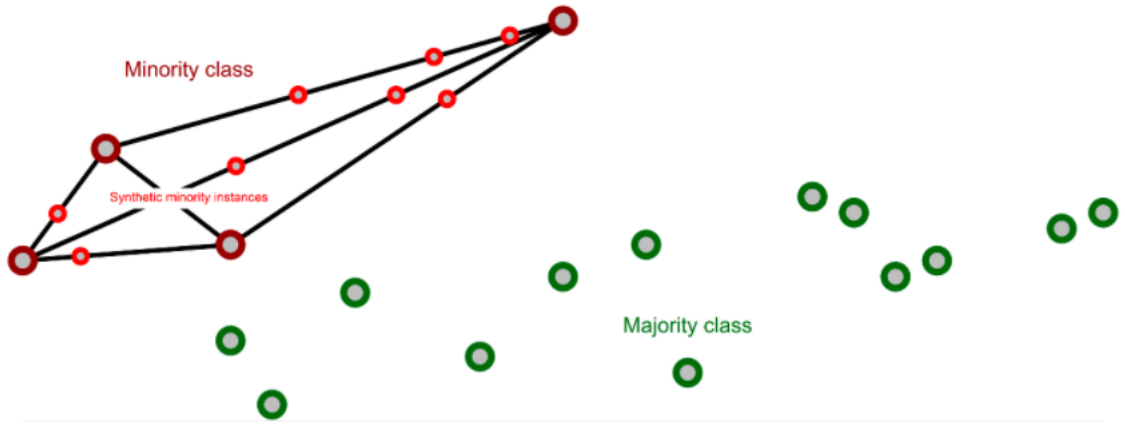

Figure 2: SMOTE OVERSAMPLING

An improvement that can be done to balance the dataset, is to implement an *Autoencoder* to encode the overall samples and use the most hidden layer that encode the input, to generate new samples.
The Autoencoder is capable of creating sparse representations of the input data, so use it, can be a better way to oversample the input dataset for balancing the data.

0.2.3.2 Standard Preprocessing

Firstly all the labels are encoded from 'Healthy' and 'Tumoral' to '0' and '1'.
Nextly the dataset is shuffled, and the train and test split is done with 20% of data for the test set.
At this point all values belonging to the samples features are scaled. The scaler choosed is the *MinMaxScaler* provided from *sklearn* library.
With this normalization all data belongs to the range [0,1], 0 for the minimum value and 1 for the maximum value in the features.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Figure 3: MinMaxScaler

Data are now ready to be fitted in a classification model.
But let's see and other kind of preprocessing.

0.2.3.3 Principal Component Analysis

This dimensionality reduction technique is applied over all the samples, and the variance between features is calculated.
Is decided to find the first 'k' *PCA Components* that have the 90% of the overall variance in the dataset.
The result is that the 90% of the variance over all the features is in the first 136 PCA components.

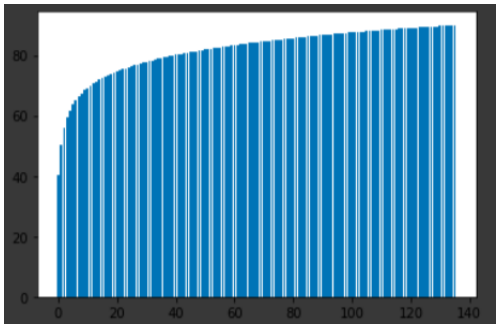

Figure 4: VARIANCE

In order to better see the data, the first 3 PCA components, that have an overall variance equal to 56.01% are plotted:

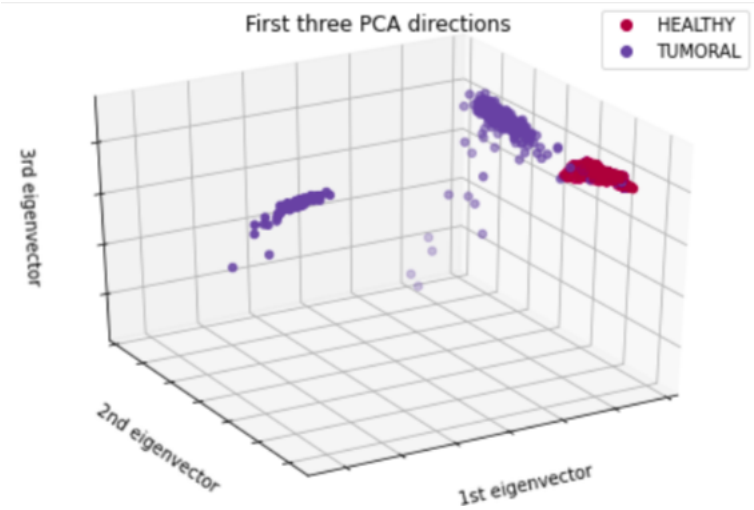

Figure 5: 3 PCA COMPONENTS

And also the 2-D transformation of the previous plot is shown below:

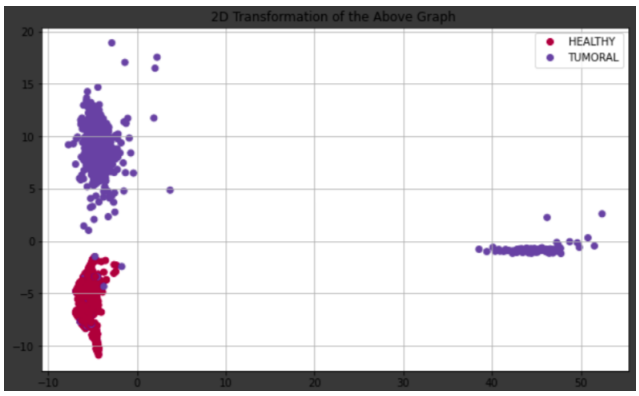

Figure 6: 2-D visualization of 3 PCA COMPONENTS

At this point the preprocessing phase is finished.
An improvement that can be done is the tuning of the type of *Standardization* used, and the number of *PCA components* to find the best trade off implementation in terms of performance and complexity of the overall system.

0.2.4 Model Definition

Several machine learning models can be used for that particular genomic classification. From the literature i had choose the following classifiers for this problem, this is done in order to see the performance of different models and find out which is the best for this particular classification problem.

- NEAREST MEAN CLASSIFIER
  This model assigns to observations the label of the class of training samples whose mean is closest to the observation.

- NEAREST NEIGHBOR CLASSIFIER
  In this model the labels are assigned based on the neighbor labels.
  K is the parameter that count how much neighbor are considered.
  Because of this classifier is a voting classifier, the number K, must be an odd number.

- NAIVE BAYES CLASSIFIER
  This is an example of *Probabilistic Classifier*.
  The model is based on the *Bayes Theorem*, so it find the *posteriori probability* based on the *likelihood probability*, the *prior probability* and the probability that sample data are observed.

- SUPPORT VECTOR MACHINE CLASSIFIER
  This classifier is non-probabilistic, binary and linear.
  It find a mathematical description of margins that divide samples belonging to different classes.
  The main parameter of this classifier is the maximum margin, so how much the samples are divided in the correct way.

- DECISION TREE CLASSIFIER
  This model create a set of nodes based on the amount of information of the features.
  In this way different branches of the tree can be followed until the classification result is performed.

- RANDOM FOREST CLASSIFIER This classifier use an *Ensemble* method to classify samples.
  The random forest classifier is made of different decision trees, and the classification is done performing a vote decision among these decision trees.

- NEURAL NETWORK CLASSIFIER
  This model is based on fully connected neural network.
  Adam optimizer is used to update the weights during the training phase.
  The learning rate is set to 1e-3.
  The number of epochs for the training are determined with an integrated early stopping technique, that monitor the validation loss, and decide when is necessary to stop the training because of the performances, that didn't improve anymore.

- BAYESIAN NEURAL NETWORK
  This model is used in the final *Consensus Classifier*.
  It is made of 2 basic blocks:

  - 1 *Fully Connected* layer of 2048 units
  - 1*Relu* activation function
  - 1*Dropout* layer in a *Variational Dropout* way with dropout rate equal to 0.1

  As in the previous neural network the *Adam* optimizer is used to update the weights during the training phase and the learning rate is set to 1e-3.
  This kind of network is analyzed when building the *Consensus Classifier*

0.2.5 Workflow

In order to see the performance on different training set the *Crossvalidation* technique is used. I decided to use this technique in a different way.
In detail all the data are shuffled and, after that, are splitted in a stratified way into training and test set.
This *Shuffle/Split* is done for all the *'k'* fold of the crossvalidation.
In this way we are not limited by simply divide data in different folds, but the training and the test set are determined by randomly shuffle the overall dataset and split it into training and test set.
So this technique is different by the classic crossvalidation, and it exploits the *random split* for train and test set.
The number of shuffles used is equal to 50.
So these results are the average of results coming from this 50 fold crossvalidation.
The performance had a very low variance on the results, around 0.1. So i decided to keep the mean of the results.

0.2.6 Results

MinMax PREPROCESSING

Models Performance with MinMax Preprocessing					
Classifier	Accuracy	Error	THRate	TTRate	TETime
Nearest Mean	95.95%	4.04%	92.67%	100%	835 ms
Nearest Neighbor	96.26%	3.72%	92.92%	100%	27813 ms
Naive Bayes	95.64%	4.35%	91.88%	100%	1348 ms
SVM	96.03%	3.97%	92.45%	100%	20386 ms
Decision Tree	94.08%	5.99%	93.87%	94.17%	17166 ms
Random Forest	94.24%	5.75%	94.09%	94.35%	17773 ms
Neural Network	96.65%	3.34%	93.73%	100%	57540 ms

1 PCA Component

Models Performance with 1 PCA Component					
Classifier	Accuracy	Error	THRate	TTRate	TETime
Nearest Mean	58.36%	41.63%	53.97%	100%	2.30 ms
Nearest Neighbor	74.47%	25.52%	72.28%	77.05%	13.12 ms
Naive Bayes	63.65%	36.34%	57.67%	95.01%	2.36 ms
SVM	76.57%	23.42%	72.08%	82.19%	40.48 ms
Decision Tree	67.16%	32.84%	67.03%	67.28%	3.47 ms
Random Forest	68.01%	31.98%	69.66%	66.851%	223.83 ms
Neural Network	78.05%	21.94%	75.89%	81.08%	818.38 ms

2 PCA Components

Models Performance with 2 PCA Component					
Classifier	Accuracy	Error	THRate	TTRate	TETime
Nearest Mean	95.33%	4.66%	91.16%	100%	2.56 ms
Nearest Neighbor	96.73%	3.26%	93.78%	100%	13.35 ms
Naive Bayes	96.18%	3.81%	93.93%	98.73%	2.72 ms
SVM	95.95%	4.04%	92.52%	100%	9.79 ms
Decision Tree	93.69%	6.30%	95.08%	92.25%	3.97 ms
Random Forest	94.24%	5.75%	93.19%	95.27%	205.16 ms
Neural Network	96.73%	3.26%	93.85%	100%	441.94 ms

3 PCA Components

Models Performance with 3 PCA Component					
Classifier	Accuracy	Error	THRate	TTRate	TETime
Nearest Mean	96.10%	3.89%	92.63%	100%	1.81 ms
Nearest Neighbor	95.17%	4.82%	91.26%	100%	13.25 ms
Naive Bayes	96.26%	3.73%	94.40%	98.37%	2.10 ms
SVM	95.87%	4.12%	91.93%	100%	10.01 ms
Decision Tree	93.23%	6.77%	94.22%	92.40%	5.03 ms
Random Forest	94.55%	5.44%	95.19%	93.92%	215.90 ms
Neural Network	97.04%	2.95%	94.39%	100%	569.68 ms

k PCA Components

Models Performance with 1 PCA Component					
Classifier	Accuracy	Error	THRate	TTRate	TETime
Nearest Mean	96.57%	3.42%	93.60%	100%	2.24 ms
Nearest Neighbor	96.80%	3.19%	93.66%	100%	15.19 ms
Naive Bayes	96.42%	3.58%	94.66%	98.49%	2.14 ms
SVM	96.65%	3.34%	93.48%	100%	10.16 ms
Decision Tree	92.76%	7.23%	93.65%	91.95%	5.20 ms
Random Forest	93.77%	6.22%	92.82%	94.65%	212.35 ms
Neural Network	96.65%	3.34%	93.76%	100%	569.68 ms

0.3 IMAGING

0.3.1 Whole Slide Image

A WSI (whole histological image) is a multiresolutional image made by digitizing microscope slides at diagnostic resolution.
They are very large images and they have multiple level of resolution.
A single WSI occupy around 1 Gbyte of memory so they are really computationally expensive to study.

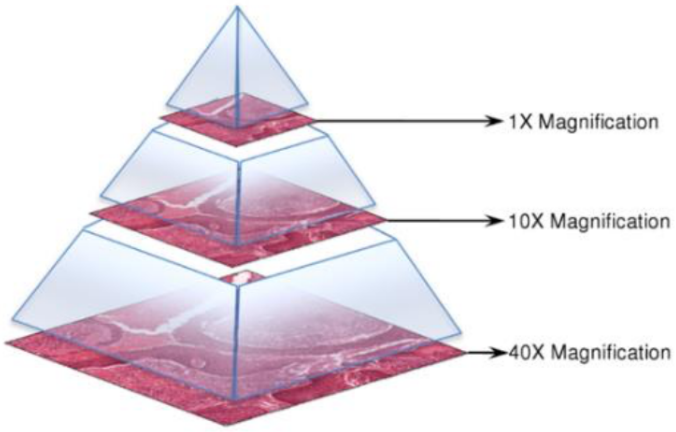

Figure 7: WHOLE SLIDE IMAGE

This resolution is typically achieved by scanning a complete microscope slide, and from these scan, create a single high-resolution image.
To do that is necessary to capturing many small high-resolution image tiles and and then mount them to create a full image of a histological section..

0.3.2 Data Preparation

As for genomics data, WSI are downloaded from GDC (Genomic Data Commons) Data Portal.
But this time the storage needed for these images, is much more larger.
The result was around 400 Gbyte. Too much.
So i decide to select the WSI for patients for which i had the genomics data.
To achieve this result several steps are done to filter out the correct *manifest* file that allows this download.
These steps are implemented with a python script that store the *case_id*'s elements wrote in the *json* file of genomics data, and try to find a match in the *json* file of the WSI data.
To make sure that this process is done in the correct way, the final manifest file is constructed copying the reference manifest file of the WSI and then delete the lines which haven't a match with *case_id*'s that i have previously stored.
So at this point the images manifest file is made of filename's that have a match in terms of case id with the genomic manifest file.
The overall storage become around 20 Gbyte.
Two examples showed below as *.png* images, healthy case on the left, and tumoral case on the right:

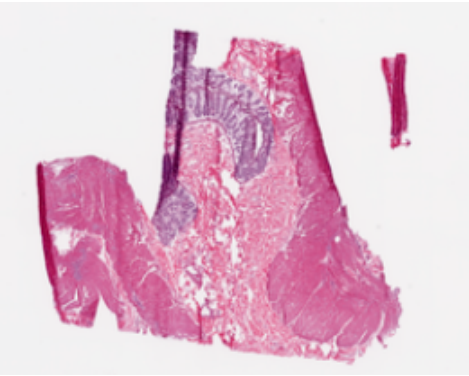

Figure 8: HEALTHY

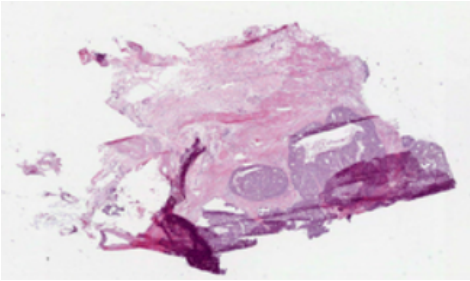

Figure 9: TUMORAL

Is possible to see that there are a lot of 'white' region without information as input data for this analysis.
The right thing to do is to crop each image in significant patches.
This is done and the overall patches are saved as png images.

0.3.3 Data Preprocessing

The preprocessing of these image dataset requires a lot o steps.
First of all significant patches are saved as *rgb* images.
At this point the dataset has to be balanced. This is done performing data augmentation with a couple of *Keras* layers, so new images are added to the dataset.
The overall dataset of 4708 patches, 2354 healthy and 2354 tumoral, is divided into training and test set with 10% of images belonging to the test data set.
The training set is also split into training and validation set with 20% of images belonging to the validation data set.
The image size and the batch size are set to 256x256 and 32 respectively.
Some patches can be seen above.

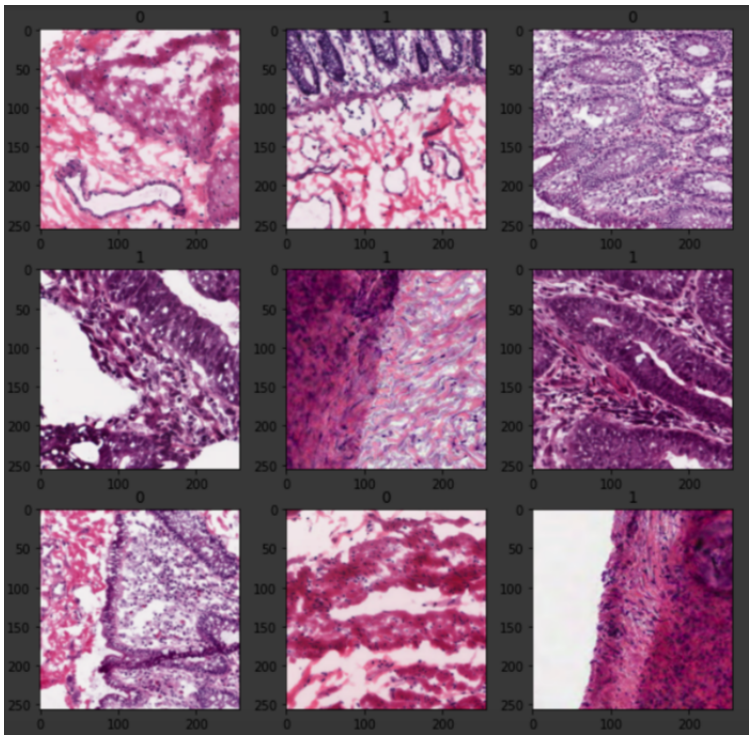

Figure 10: RANDOM PATCHES

In order to try to unbias as much as possible the models implemented later , a couple of *Keras* preprocessing layers are put at the beginning of the model in order to augment data also runtime during the training phase, and also during inference.

The preprocessing layers used are listed below:

- Random Horizontal Flip
- Random Rotation

Also the rescaling of input data is done inside the model with a rescaling preprocessing layer.

0.3.4 Benchmark Models

Some benchmark models as *Xception* and *ResNet* are implemented in order to have a reference in terms of performance.
The main characteristics of these two networks are highlighted below:

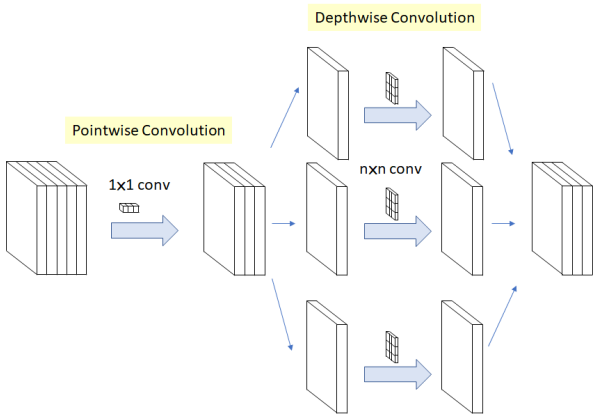

Figure 11: XCEPTION MODULE

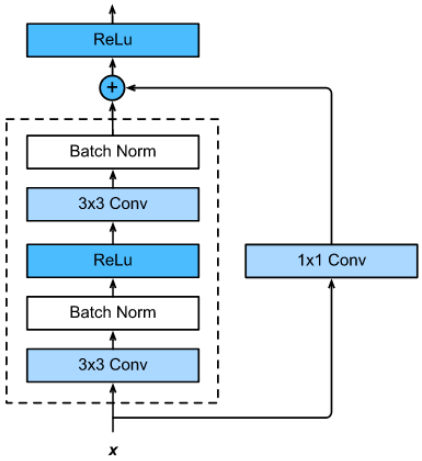

Figure 12: RESNET MODULE

These models are trained from scratch, in a *Transfer Learning* updating just the weights in the classification layer during training.
Also the *Fine Tuning* is performed to update also the weights inside the hidden layers, with a very low learning rate to avoid overfitting.
The classification layer is a fully connected layer is always implemented with 1 Dense layer of 1024 units and *ReLu* activation function, followed by a Dropout layer with 0.5 rate.

0.3.5 Workflow

After a long hyperparameter tuning the value of them is decided to be as following:

- The optimizer decided to use for the backpropagation algorithm is *Adam*.
- The learning rate is set to 5e-4.
- Number of epochs: 40.
- Metrics monitored during training and test:
  - Binary Crossentropy
  - Binary Accuracy

  - False Positive
  - False Negative
  - True Positive
  - True Negative
  - Precision
  - Recall

- Early Stopping with the following features:
  - Parameter monitored: *Validation Loss* .
  - Number of epochs to be patient: 15 .

- Learning Rate reducing with the following features:
  - Parameter monitored: *Validation Loss* .
  - Number of epochs to be patient: 5 .
  - Factor by which the learning rate is multiplied: 0.5 .

The last two points are callbacks added when performing the training.
In this way the *validation loss* is monitored during training.
With the first callback is decided to stop the training after no improving of the loss function in the validation dataset for 15 epochs.
The second callback is related to reducing the learning rate during the training when the loss function has no improvement in the validation set for 5 epochs.
These techniques shows a big improvement in terms of avoid overfitting and improving performances.

0.3.6 Results

First of all i decided to perform the *Transfer Learning* from Xception network.
The results are shown below:

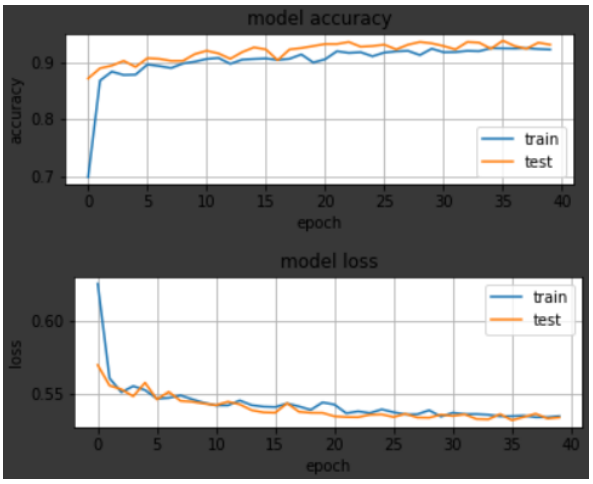

Figure 13: XCEPTION TRANSFER LEARNING TRAINING PARAMETER

```
Loss:            0.5965609550476074
Accuracy:        0.7510638236999512
False Positive:  0.0
False Negative:  117.0
True Positive:   118.0
True Negative:   235.0
Precision    :   1.0
Recall:          0.5021276473999023
```

Figure 14: XCEPTION TRANSFER LEARNING RESULTS ON TEST SET

After the Transfer learning also the *Fine Tuning* is exploited.
The learning rate in this case is very small, because all the parameters are updated, and to avoid overfitting, this main parameter has to be very small. I choose 5e-6.
The results of applying also the fine tuning to this model are shown below:

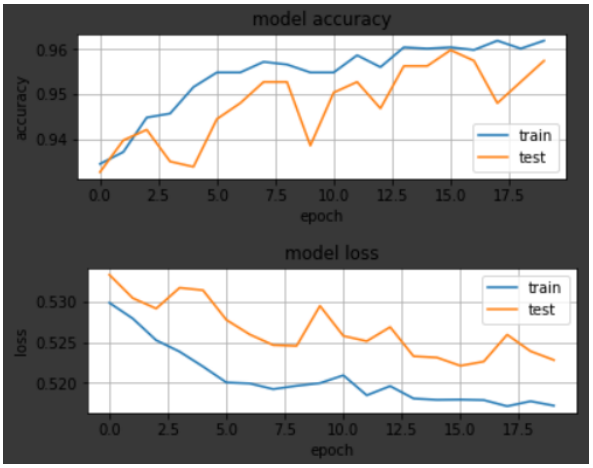

Figure 15: XCEPTION FINE TUNING TRAINING PARAMETER

```
Loss:            0.5767318606376648
Accuracy:        0.8042553067207336
False Positive:  0.0
False Negative:  92.0
True Positive:   143.0
True Negative:   235.0
Precision    :   1.0
Recall:          0.6085106134414673
```

Figure 16: XCEPTION FINE TUNING RESULTS ON TEST SET

As expected the Fine Tuning improves the model performances a little bit, but is possible to see that over the 10th epoch seems that the model is going to overfitting.
The previous steps are replicated also with the *ResNet-152* network.

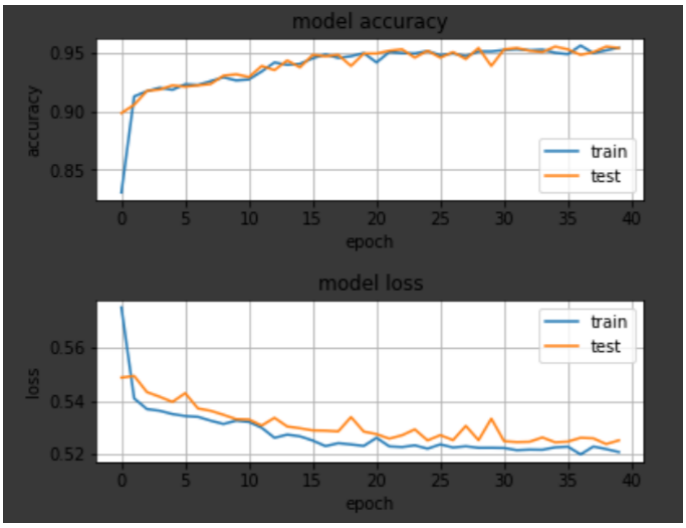

Figure 17: RESNET TRANSFER LEARNING TRAINING PARAMETER

```
Loss:            0.5825520157814026
Accuracy:        0.7914893627166748
False Positive:  1.0
False Negative:  97.0
True Positive:   138.0
True Negative:   234.0
Precision    :   0.9928057789802551
Recall:          0.5872340202331543
```

Figure 18: RESNET TRANSFER LEARNING RESULTS ON TEST SET

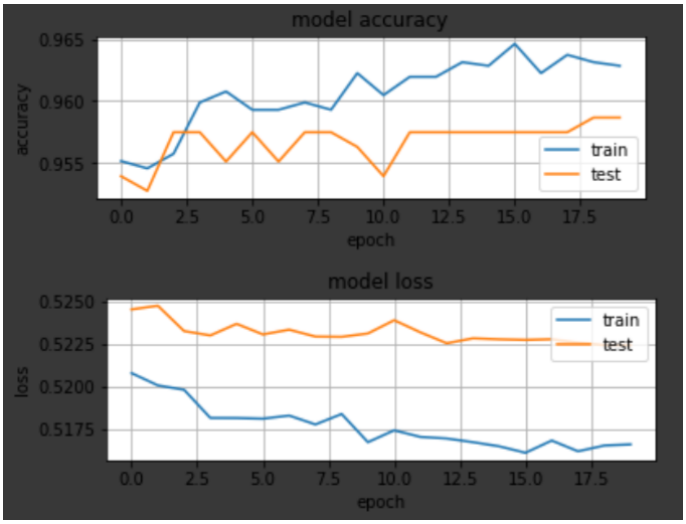

Figure 19: RESNET TRANSFER LEARNING TRAINING PARAMETER

```
Loss:            0.5810348391532898
Accuracy:        0.7957446575164795
False Positive:  0.0
False Negative:  96.0
True Positive:   139.0
True Negative:   235.0
Precision    :   1.0
Recall:          0.5914893746376038
```

Figure 20: RESNET TRANSFER LEARNING RESULTS ON TEST SET

In the ResNet model is possible to see that the fine tuning achieves the overfitting and the validation accuracy didn't improve anymore.
Probably if the the training continue, the validation accuracy may decrease due to overfitting.
Nextly in order to see the correct trend of training and validation parameters, such as accuracy and loss, in a Deep Neural Network model, the Xception model is loaded

without including the imagenet weights and the top layer (Classification layer).
So a correct trend of training parameter can be seen with a benchmark model.
The model is trained for 30 epochs.
The learning rate is equal to 5e-4.
The *Early Stopping* and the *Reduce Learning Rate On Plateau* are set as before.
This are the results:

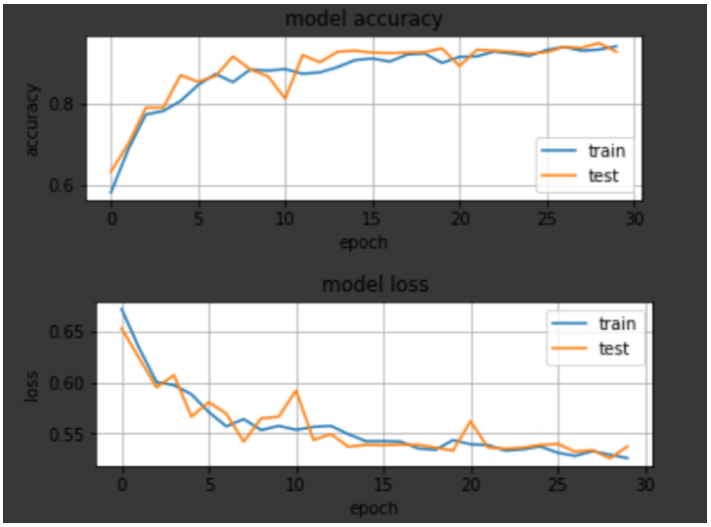

Figure 21: XCEPTION TRAINING FROM SCRATCH PARAMETER

```
Loss:              0.5648940801620483
Accuracy:          0.8404255509376526
False Positive:    2.0
False Negative:    73.0
True Positive:     162.0
True Negative:     233.0
Precision    :     0.9878048896789551
Recall:            0.6893616914749146
```

Figure 22: XCEPTION FROM SCRATCH RESULTS ON TEST SET

As is possible to see the performances has improved compared to the Transfer Learning and the Fine Tuning when we had load the *imagenet* weights.
This proofs that for different image classification problems, the weights of the model, may be different in order to shows the best performance for that particular classification problem.

0.3.7 My Model

At this point, with diffent reference performances, i tried to implement my own model. I want to exploit different tricks that has shown good results, like the residual module of the Resnet network, and the deepwise and pointwise module of the Xception module.

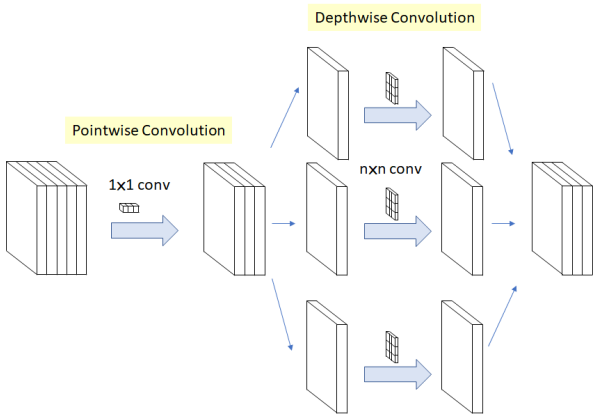

Figure 23: XCEPTION MODULE

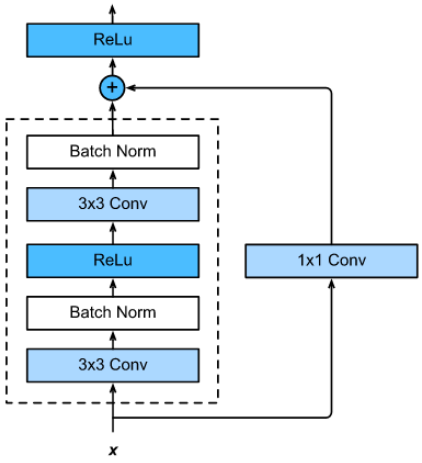

Figure 24: RESNET MODULE

I also want to make my model as a *Bayesian Neural Network Model*.
To do that the technique that implement this kind of network is the *Variational Dropout*, that exploit the answer in terms of probability, and this is given not just by the weights, but is given in terms of mean and variance of the of that weights.

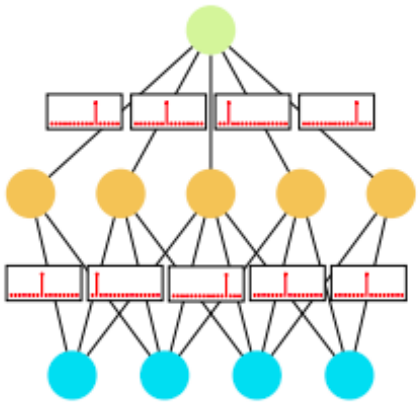

Figure 25

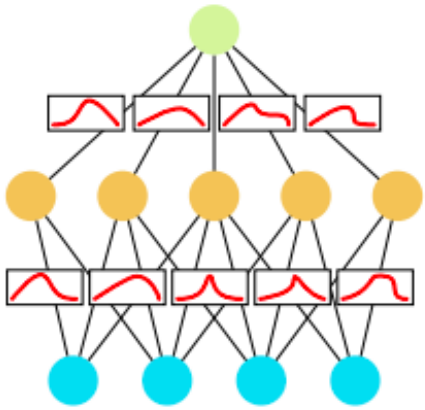

Figure 26

The built model is very deep but i tried to show it in next 3 pages:

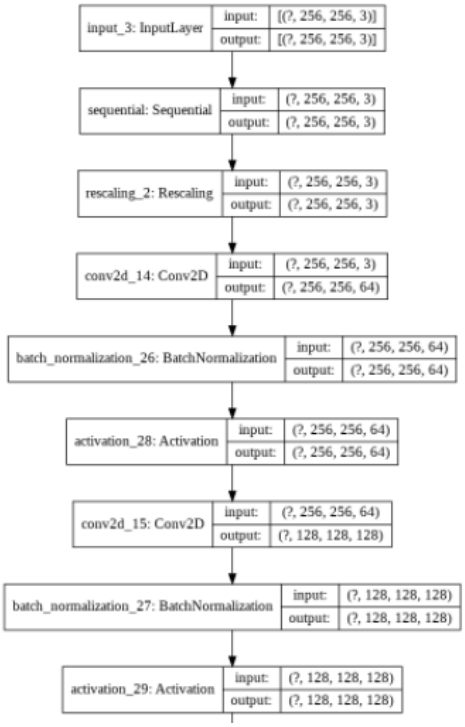

Figure 27: MY MODEL

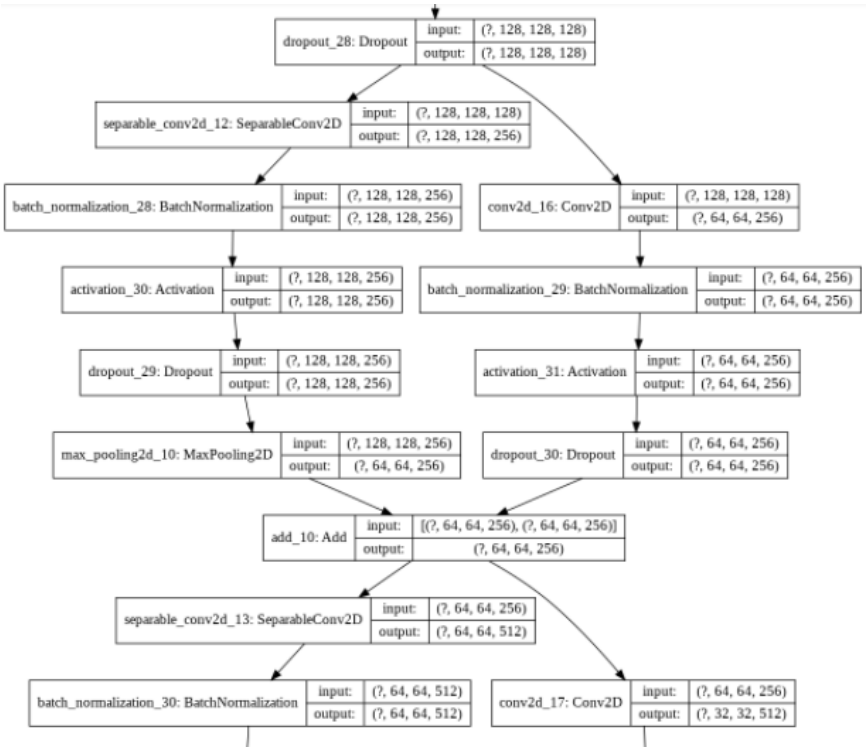

Figure 28: MY MODEL

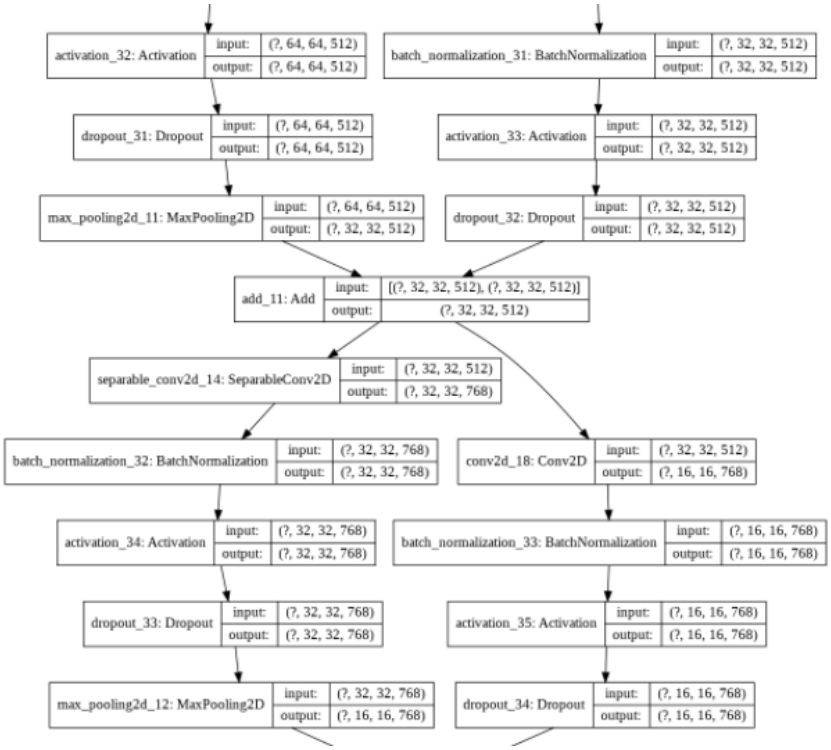

Figure 29: MY MODEL

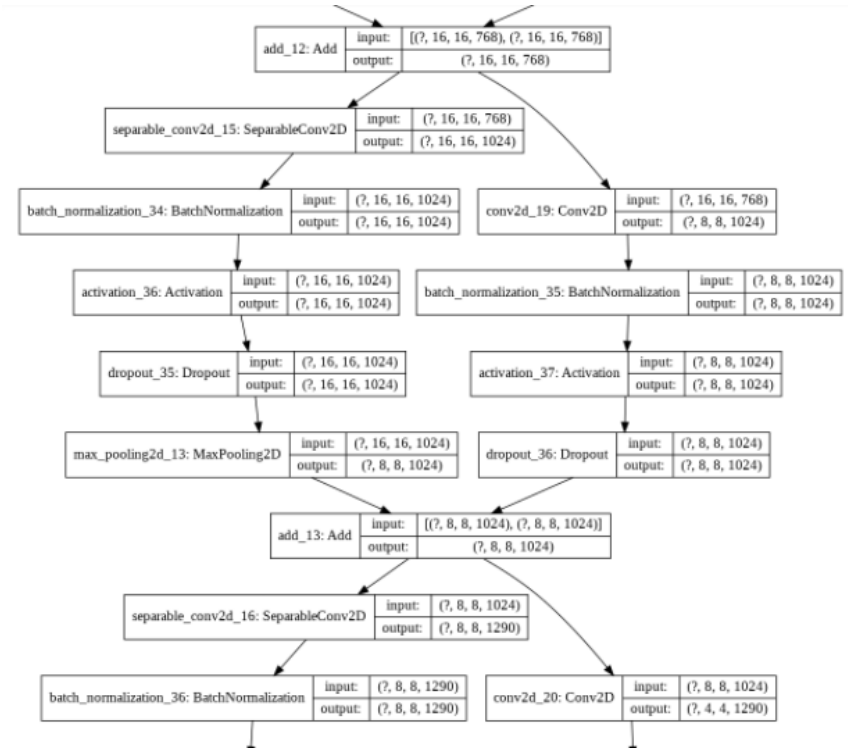

Figure 30: MY MODEL

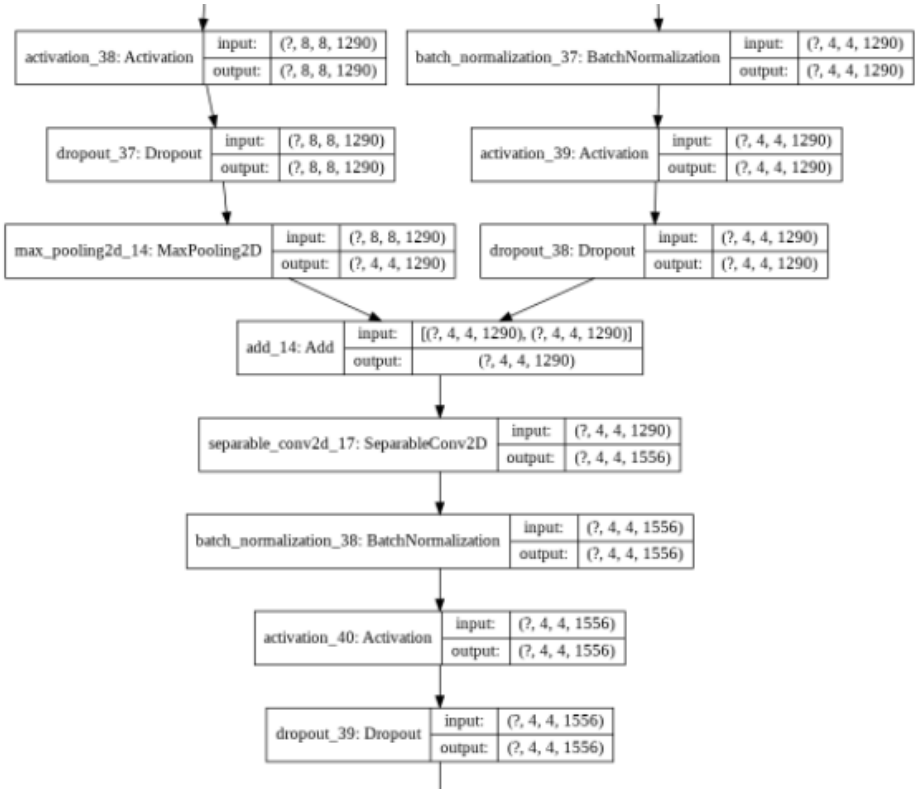

Figure 31: MY MODEL

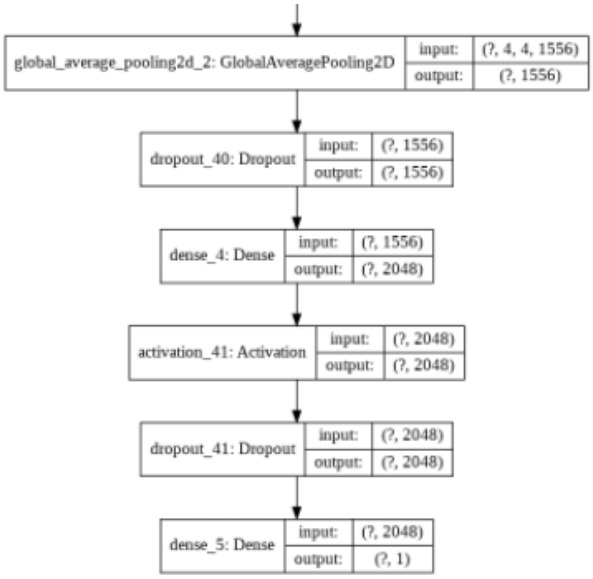

Figure 32: MY MODEL

The compiling parameters are the following:

- The optimizer decided to use for the backpropagation algorithm is *Adam*.
- The learning rate is set to 1e-4.
- Number of epochs: 40.
- Metrics monitored during training and test:
  - Binary Crossentropy
  - Binary Accuracy
  - False Positive
  - False Negative
  - True Positive
  - True Negative
  - Precision
  - Recall
- Early Stopping with the following features:
  - Parameter monitored: *Validation Loss* .
  - Number of epochs to be patient: 20 .
- Learning Rate reducing with the following features:
  - Parameter monitored: *Validation Loss* .
  - Number of epochs to be patient: 5 .
  - Factor by which the learning rate is multiplied: 0.5 .
- Dropout rate equal to 0.1.

0.3.8 Results

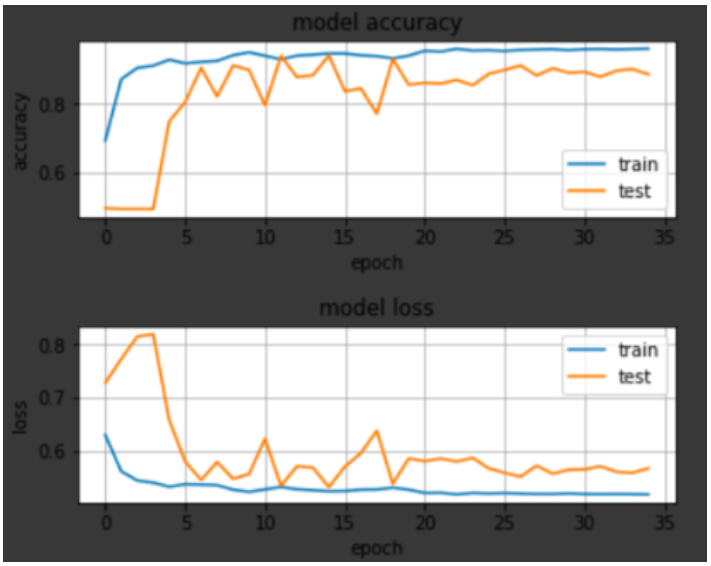

Figure 33: MY MODEL TRAINING PARAMETER

Loss:	0.5712559223175049
Accuracy:	0.8361701965332031
False Positive:	11.0
False Negative:	66.0
True Positive:	169.0
True Negative:	224.0
Precision :	0.9388889074325562
Recall:	0.7191489338874817

Figure 34: MY MODEL RESULTS ON TEST SET

From these results is possible to see that the Early stopping callback has occur at the 34th epoch.
The Learning Rate was reduced 4 times before that stop on the training.
The validation loss and accuracy show some fluctuation in the first 20 epochs, and only after that is almost set to the same value.
This is probably due to the *Variational Dropout* layers that takes some time before learning on the different selected weights during training.

0.4 CONSENSUS CLASSIFIER

The purpose of this step is to integrate the knowledge derived from different domain of data classification:

- Genomics data classification
- Whole Slide image classification

The idea is to build a *Bayesian Ensemble Classifier* that works with heterogenous data. The genomics data classification shows their best performance with the neural network classifier.
So is possible to modify this network in order to be bayesian, adding dropout layers in a *Variational Dropout* way.
This leads to have results in terms of mean and variance of the classification.
If we use an odd number of these classifier the results are an odd number of mean and variance classification.
So at this point we have different classifiers, that acts in a different way because of the variational dropout.
If we think to add now, an even number of bayesian classifier, to classify the Whole Slide Images, at the end we have an odd number of bayesian classifiers that act on different domains of interest, genomics and images classification. But the answer of these classifiers is always a mean and a variance of this answer.
I thought about a formula that weights the answer(mean) based on its variance over all classifiers.
When we extract the output value of the *Sigmoid* activation layer which is between 0 and 1, the mean and the uncertainty is calculated after N iteration where the model is used in *Inference*.
Let's think about some examples, if we get a $\mu = 0.99$ and $\sigma = 1e - 7$, this means that the classifier is almost sure about its answer, and it means that the classification answer belongs to healthy. Instead of an answer with a big σ, that shows uncertainty about the classification.
What we want to have as result is simple:
The maximum uncertainty of the correct predictions has to be less then the minimum uncertainty over the wrong predictions.
The problem is that, the main assumption is that the classifiers are able to classify with low uncertainty when the prediction is correct and with high uncertainty when the prediction is wrong.
This is achievable if, for the correct predictions, the majority of classifiers predict that label, and, for the wrong prediction, there is a sort of balancing in the predictions, so that it can be classified as uncertain prediction. And for binary classification, if this happen, is known that the correct prediction is the opposite label, so applying a *Reverse label prediction based on uncertainty* may correct the wrong answers.

This can leads to determine some bounds in terms of uncertainty in order to define if the predictions are certain or uncertain, it can be called $\sigma_{threshold}$, and when the prediction belongs to uncertain, predict the opposite label because of the binary classification.
Ideally, if this happen, the accuracy of a binary classification can achieve 100%, and obviously can be extended to multilabels classification.
But let's try to see the real results of this study.
In order to do properly this kind of inference, i had to use the *predict* method to have the output of the sigmoid activation function of the last layer, and round that value to 1 or 0. This prediction is done several times to have a mean and a variance of that prediction over the same input.
The problem is that with the *predict* method instead of the *evaluate* method, the performances are not the same on the Image Classifier.
In particular is possible to see a sort of bias towards the healthy label.

The results are shown below:

- Genomics
  For this case the different 5 different models are implemented as Bayesian Neural Network:
  - 1 dense layer with 4 unit
  - 1 dense layer with 8 unit
  - 1 dense layer with 16 unit
  - 1 dense layer with 64 unit
  - 1 dense layer with 128 unit

  All these networks are followed by a *ReLu* activation function, a dropout layer with dropout rate equal to 0.1 and the finale single unit dense layer with *Sigmoid* activation function.
  The optimizer chosen is *Adam* as in all previous network, with learning rate equal to 1e-4.
  In this case the *Reverse label prediction based on uncertainty* technique has shown a little improvement.
  The new hyperparameter $\sigma_{threshold}$ is equal to 0.249.
  After the predictions the accuracy was equal to 100% for healthy cases and 95.29% for tumoral cases.
  After applying the technique explained previously, with the accuracy for the healthy cases remain as before, and for tumoral cases it has improved to 95.72%.

- Whole Slide Images
  For this case the 3 bests models are used and they show almost the same performances.
  The networks are like the Image Classification Model previously presented.
  The difference is just on the fully connected layer, which is a dense layer with 2048 units, or two dense layer with 1024 units, otherwise directly the dense layer with the sigmoid activation.
  When evaluate these models, the overall accuracy is equal to 81.05%. So using this classifier, simply slow down the performances of the *Bayesian Ensemble Classifier*.
  There is one more problem that i tried to solve, but without results:
  Because of the problem previously explained on *predict* and *evaluate* methods, i can't make the same study based on uncertainty as in the genomics model.
  I tried several times but when predict and round the output of the sigmoid activation, the performances are around 50% and i haven't found out an answer for that reason.
  But, even if i will be able to do that, the performances slow down because of the poor accuracy of the WSI classification.

The inferences for genomics and images are done separately because of their different structure and so their different behavior.
The overall accuracy can't be calculated because of the problem previously explained.
But some new knowledge comes out from this study.
In particular that the *Reverse Label based on Uncertainty* can improve the performances of a classifier.
The key of this approach is the tuning of the new hyperparameter $\sigma_{threshold}$.
As conclusion, this case study, where the same tumoral and healthy tissues are analyzed in two different ways, based on *Gene Expression* and on *Histological Images*, tell us that there is no classifier that works with best performances on heterogenous data, but is necessary to build good performance classifier for the available data, and see which of them works better.
In order to build a *Consensus Classifier* that works well, is necessary to build at least one classifier for each kind of data, either with best possible performance.
This gives an answer in terms of which are the data that contain the most relevant information for a correct classification.
And a consensus classifier, simply bring down the performances due to less performance in one data domain.
At the end this study proofs that the *Genomic* analysis works better than the *Whole Slide Image* analysis.

0.5 References

https://www.ncbi.nlm.nih.gov/probe/docs/applexpression/
https://scikit-learn.org/stable/
https://rikunert.com/SMOTE_explained
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4792409/
https://keras.io/guides/
https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568
https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035
https://arxiv.org/pdf/2007.06823.pdf