CLASSIFYING KEYFRAME IMAGES USING DEEP LEARNING

(With PyTorch)

1. Data Visualization

Script : visualization.py

The npz files were loaded with 'load' module of numpy package.

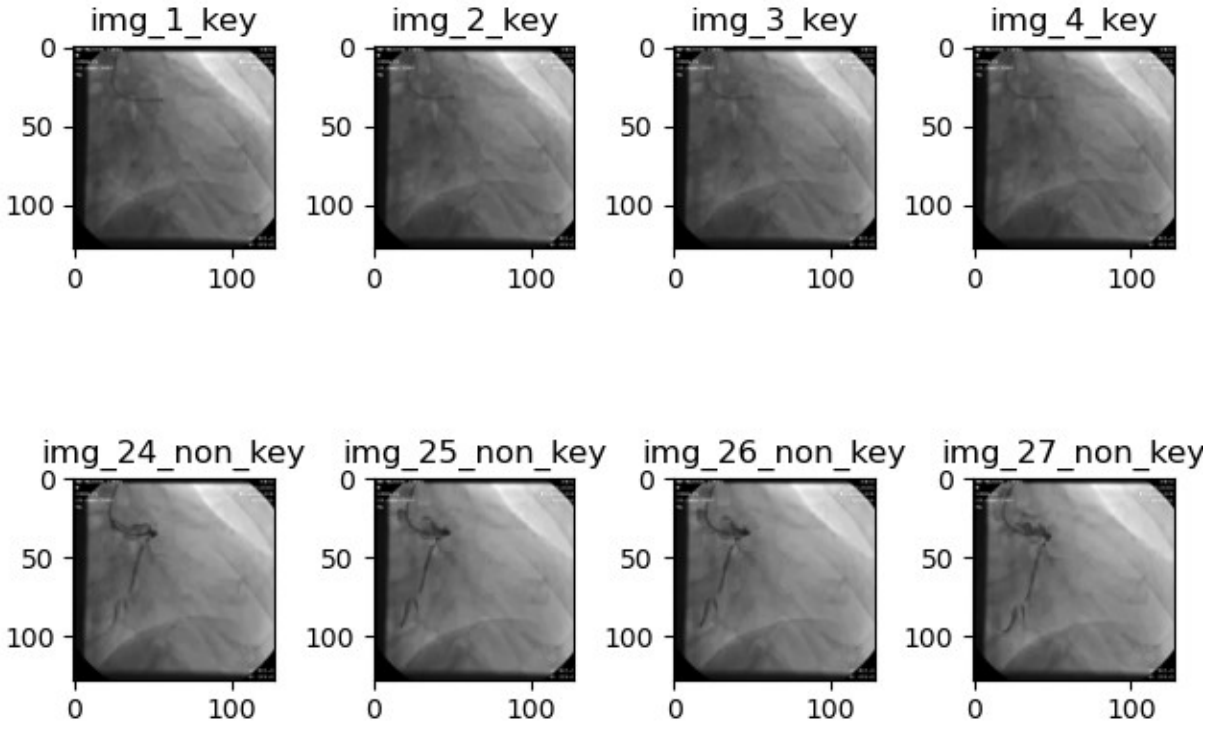
The npz-1.npz file contain a single array representing the 126 black and white images of pixel dimension 128X128.

The labels files are opened with **read_csv** module of **pandas library**.

There are two classes of labels- '0' and '1'.

The individual images or set of images can be visualized by plotting the pixel data using **matplot library**.

Sample Images:



2. Preparing first model: Model 1

Script : mod1.py

i) Loading and observing the data

The npz files and label csv files were opened as described previously. The five npz files images were extracted as numpy arrays as a1:a5 and their respective annotations as lab1:lab5.

Npz-1 to npz-5 contain 126, 136, 174, 129, 129 B/W images of dimension 128X128

ii) Dividing the train-test datasets

To start with, an unbiased model with equal number of 0 and 1 class points was decided to be prepared.

The entire data was combined in all_points numpy (with labels = all_lab).

Out of total 694 images,

300 belong to class '1'

394 belong to class '0'

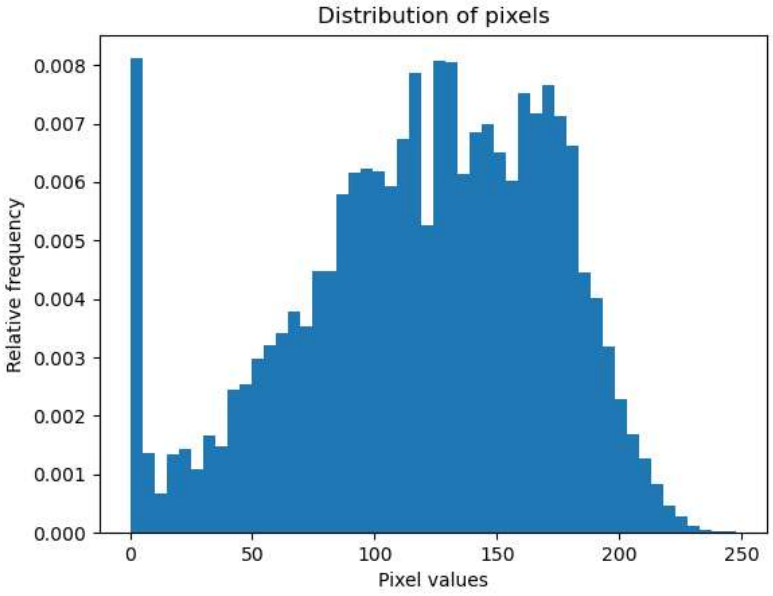
To have equal proportion of the two classes points, 300 points of class '1' were randomly picked while all the points of class '0' were used.

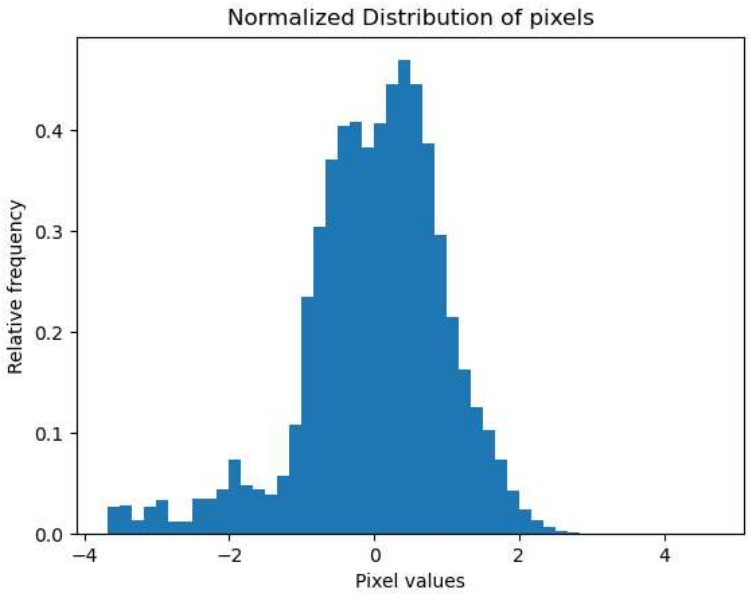
The class '0' and class '1' points were then divided in the 4:1 ratio as train:test sets.

iii) Data Normalization

The raw pixel data of the training set was visualized using matplot lib.
To reduce the pixel data values (0-255) to lower values and make it zero centered, normalization was performed.
For each image
Normalized image pixels = (pixel – mean[of 128X128 pixels])/std [of 128X128 pixels]

The normalized pixel data is re-plotted.





As can be seen from the second figure, the data becomes then zero-centered and follows more of the normal distribution.

iv) Creating Dataset class and loading the data

Class 'My_dataset' was created to convert the train and test data into tensors with of pixel values and labels

The two datasets were the loaded using torch utility **DataLoader** with batch size=40 for batch enumeration.

v) Defining the model

A Convolutional Neural Network (CNN) was built with 4 layers:

First layer: consists of

One convolutional layers with kernel size of 3X3 to reduce features

Activation function (Rectified Linear)

Dropout (with probability of 10% is applied to drop out features)

Max. pooling layer

Second layer: is similar to first layer

Third layer: consists of

fully connected layer

activation function

Output layer: consists of

Linear layer

Activation function (Softmax function to convert output to probabilities predicted for two classes

These layers were applied in the same sequence to forward propagate the input.

vi) A train_model function was written to write the flow of steps to train the model from applying the function CNN to input pixel data, calculating the loss using Criterion function to optimizing using SGD function with learning parameter=0.00003 during back-propagation. The losses on training were stored in list.

vii) To evaluate the model on test set, evaluate_ model function was written to apply the trained CNN to test data points, store the predictions (in terms of both predicted labels and single score: to represent probability of predicted class to be '1'), calculate accuracy of the model.

viii) The model is then trained and evaluated on test data.

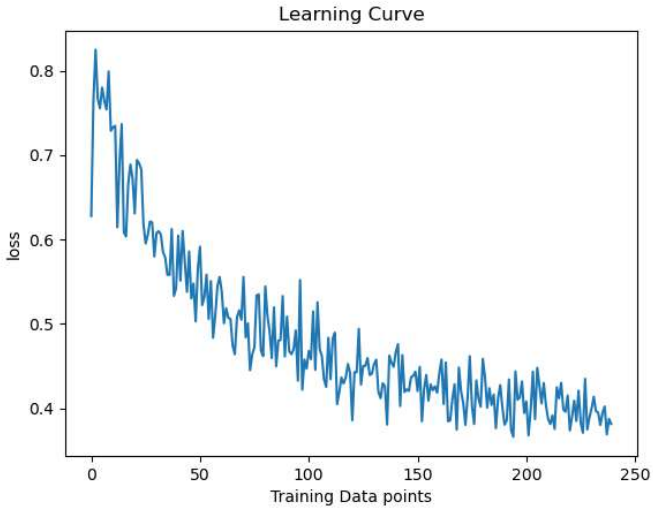
ix) The learning curve is drawn.

x) The tpr, fpr, precision, recall were calculated from predicted single score using metrics library of sklearn and ROC curve and PR curve were drawn.

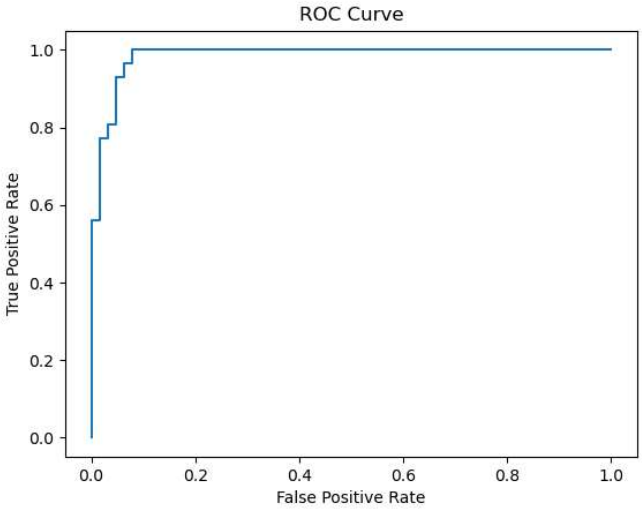
Results:

To optimize the model, different values of learning rates were tried. The final model with learning rate=0.00003 showed the best performance.

The models showed an accuracy of 0.95.

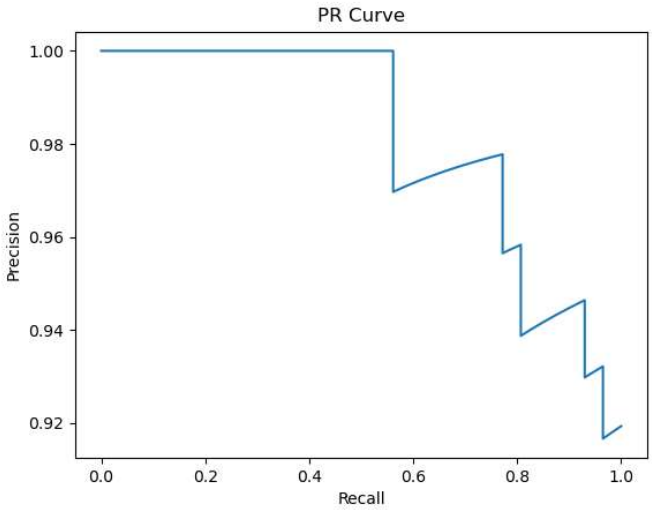


The learning curve shows the reducing loss with increasing number of training data points and shows training saturation after learning with 150 data points. This shows that the model is sufficiently trained.



The ROC curve shows an Area Under the Curve (AUC) value of 0.98 showing good performance.

The model can identify the keyframes with a True positive rate (TPR) of over 90% with 10% False positive rate (FPR).



The precision-recall curve shows approx. 94% precision at 80% TPR on the test data.

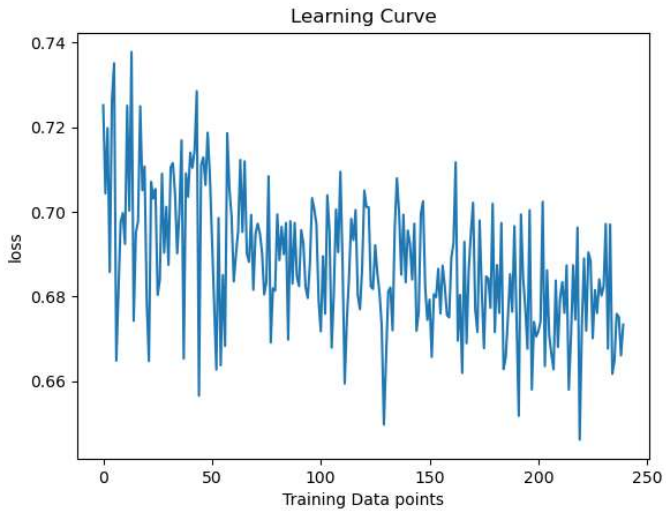
The PRC-AUC is 0.98

3. Preparing second model: Model 2

Script : mod2.py

After standardizing the **Model 1** with 2 convolutional layers, training a second model was tried with more number of layers.
This model's structure is same as model1 but **two more hidden layers** (convolutional layers) **were added** (for details see mod2.py script).

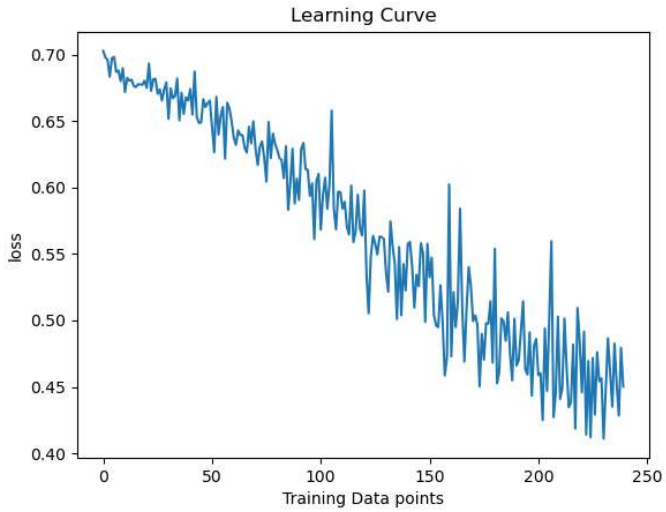
On adding more number of hidden layers, the model's performance reduced drastically (**accuracy=0.65**). Infact, it had lost it's learning ability as can be inferred from the learning curve below.



The **probable reason** for this could be that increasing the number of layers made the network more complex and now it requires more learning to be performed (more epochs) to achieve the same learning as achieved with less number of layers.

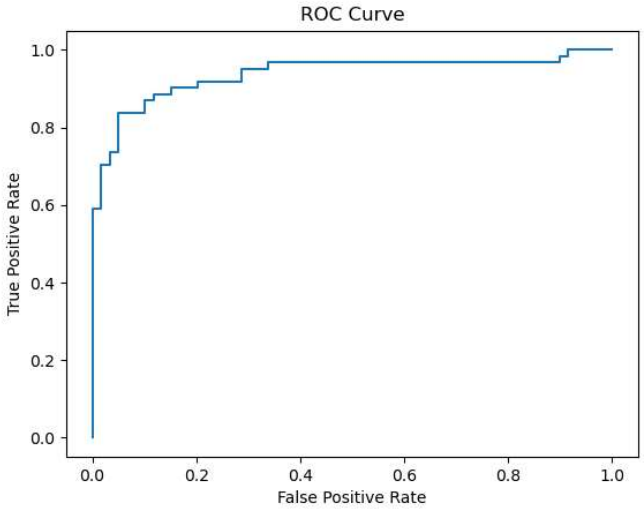
Adam optimizer is said to perform better with long networks. So we changed the optimizer **from SGD to Adam (learning rate kept the same, value=0.00003)** and now the model is again trained.

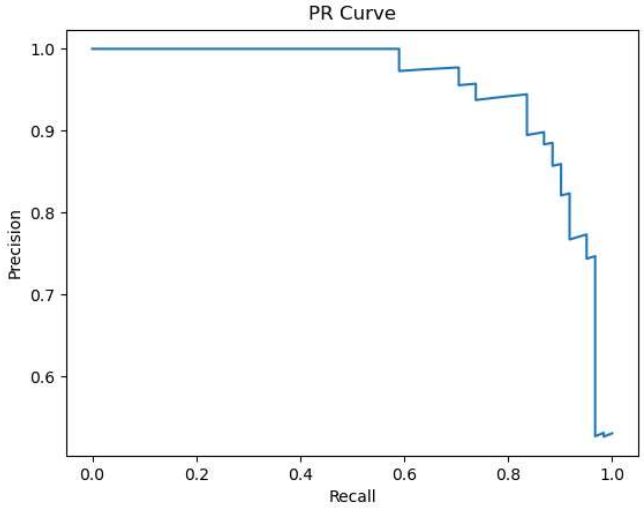
The model showed much improvement and showed an **accuracy of 0.88**. The improvement is reflected in the learning curve as well.



Form the learning curve, we infer now the loss value is decreasing with more training, but it hasn't achieved saturation yet. Therefore, Model1 seems to the better model out of the two**.**

The ROC and PR curve of the second model with Adam optimizer are shown below.
The AUC of the ROC and PR curve are 0.94, 0.98 respectively.





4. Preparing third model: Model 3 (with Blind Testing)

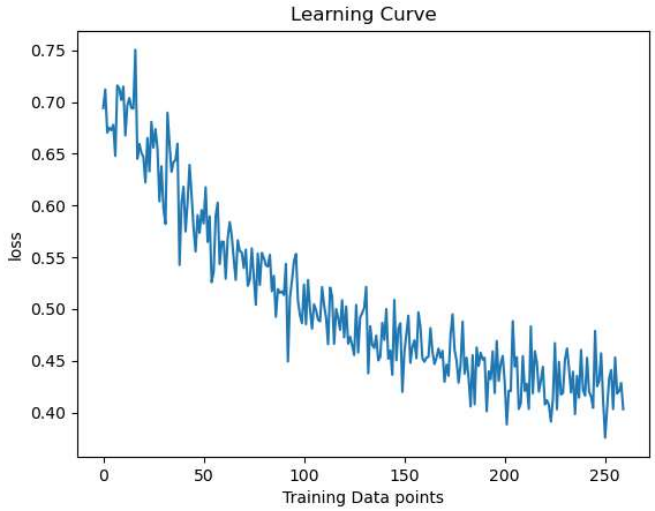
Script: mod3.py

The **Model1** showed good accuracy on test data where the test data was randomly selected (20%) from the combined set of all data.

On visualizing the datasets from the 5 dataset files, we find some patterns common among the same file images as compared to across the 5 files. So to perform a **blind testing**, **one of the data file (let's say npz-3) was kept separate for testing and the remaining 4 npz files data was clubbed and used for training**.

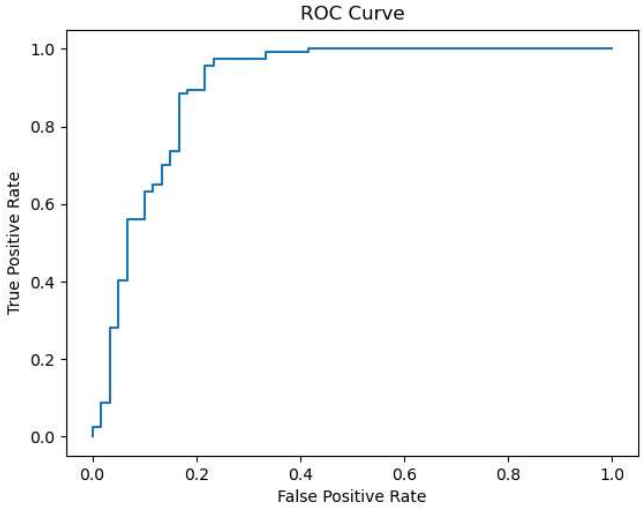
The model architecture was kept same as Model1.

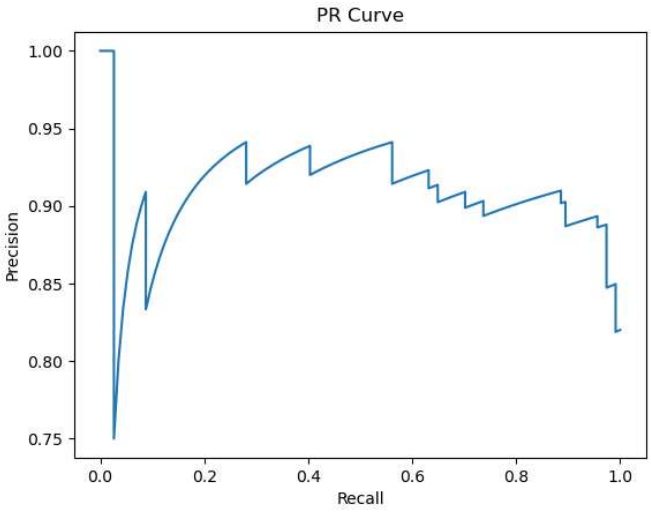
Results:



The learning curve is similar to one obtained in Model1 training, with saturation achieved after training with 160 data points.

In case of this blind testing, the accuracy achieved is 0.87.
The ROC and PR curve show AUC values of 0.90 and 0.91 resp.
The model shows slightly reduced performance (accuracy from 0.95 to 0.87) on independent test data as compared to randomly selected test data (Model 1) which was more like K-fold validation data.
Neverthless, the model is able to identify the keyframes with good accuracy in unknown data also. This Model3 shows the true predictive power of the model which shows an accuracy of 0.87 after considering all the genuine variations in the keyframes which can arise in keyframe images based on independent situations of data capturing.


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


PR Curve
Precision
Recall
0.75
0.80
0.85
0.90
0.95
1.00
0.0
0.2
0.4
0.6
0.8
1.0