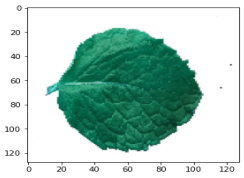
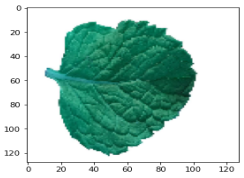
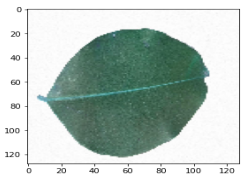
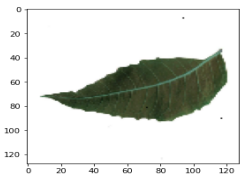
Medicinal Plant Leaf Classification using Deep Learning

A brief outline of the key features and steps involved in the development of this project is as:-

- **Dataset Description** -
    - The dataset consists of 1500 image samples of thirty species of healthy medicinal herbs. The image samples have been observed to be clearly visible and the captured pictures are relatively similar in the aspect of the positioning of the device to capture the image and the visibility of that can be used for building intelligent systems for the recognition of medicinal plants. The dataset contains high-quality images of healthy and mature leaves of different plants of the same species. The image samples are clean and hence do not need any procedures of augmentation on an initial preprocessing before the task of learning appropriate features of classification.
    - The dataset comprises thirty species of healthy medicinal herbs such as Santalum album (Sandalwood), Muntingia calabura(Jamaica cherry), Plectranthus Amboinicus / Coleus Amboinicus (Indian Mint, Mexican mint), Brassica Juncea (Oriental mustard),and many more. The goal is to automate the procedure of the best feature extraction which is the most appropriate for identifying the image pixels for classification.
- **Data Visualization**
    - The data distribution across different categories is visualized by using libraries as ***matplotlib*** and ***seaborn***.
    - It is ensured that the split in the Train, Validation and Test Set is stratified across the three datasets.
    - The split ratio is 60:20:20 for Train, Validation and Test Set respectively.
    - A clear class imbalance is observed amongst different sets and certain categories as Indian Mustard, Hibiscus Rosa-Sinensis, Nyctanthes Arbor-tristis (Parijata), Syzygium Cumini (Jamun) and Trigonella Foenum-Graceum (Fenugreek) are relatively lower in count.
    - Four randomly visualized Data Samples are as follows

- ## Data Pre-Processing
    - The given images are resized to the dimensions as (128,128,3).
    - Each pixel value for the given pixel intensity values in the array of pixels is divided by the value of /255. for normalization.
    - The batch size is kept at 32 and the images are processed in the size of batches of size 32 each.
- ## Data Augmentation
    - Augmentation of the data was done using the below 4 broader categories. Four new datasets were created for the below four categories:-
        - **Contrast based**

            Random Contrast

            Random Brightness

            RGB Shift

            JPEG Compression
        - **Flip based**

            Horizontal Flip

            Vertical Flip

            Random Rotated

            Random Crop
        - **Nature based**

            Random Sun-Flare

            Random Fog

            Random Snow
        - **Noise based**

            IAAAdditiveGaussianNoise

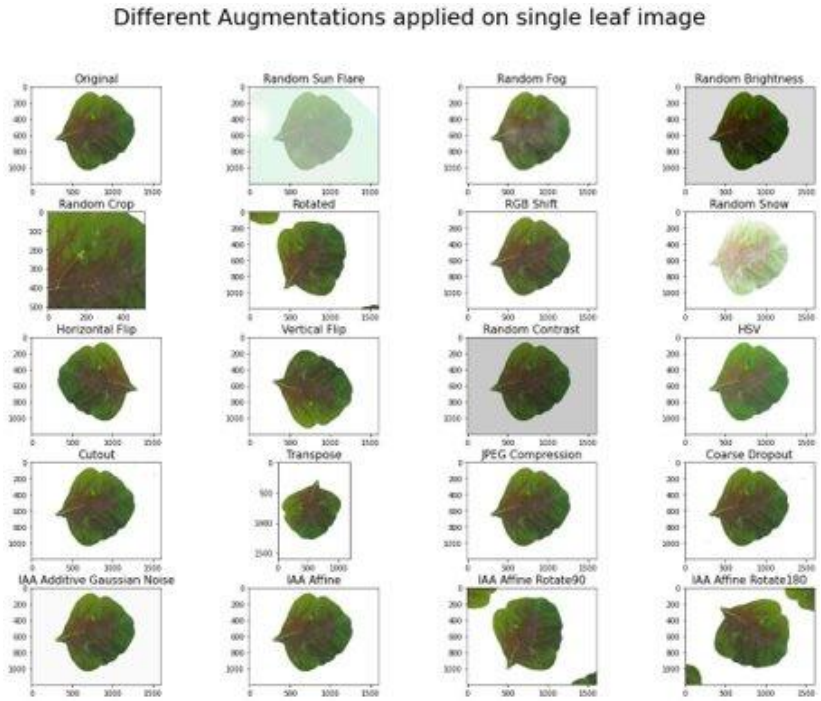
            IAAAffine (90 degrees)

            IAAAffine2 (180 degrees)

            IAAAffine3 (reflect_mode)

    - In each category, four distinct augmentations were applied. The whole dataset was split into equal parts according to the number of augmentations in the class, and one augmentation is applied to each chunk.
    - For each of the categories, after distinct experimentations, only the most relevant augmentations were selected finally for the purpose of generating augmented data samples for enhanced training of individual models.

- The distinct augmentations can be visualized as follows: -



● Deep Learning Network Architectures

○ ResNet50

- RESNET-50 is a Convolutional Neural Network with 50 layers
- It uses residual blocks, which allow information to flow more easily through the network by skipping over some layers.
- ResNet-50 uses batch normalization, which helps to stabilize the network during training and can speed up convergence.
- ResNet-50 uses a 7x7 Convolutional Layer with a stride of 2 as the first layer, which reduces the resolution of the input image and increases the receptive field of the network.

○ InceptionV3

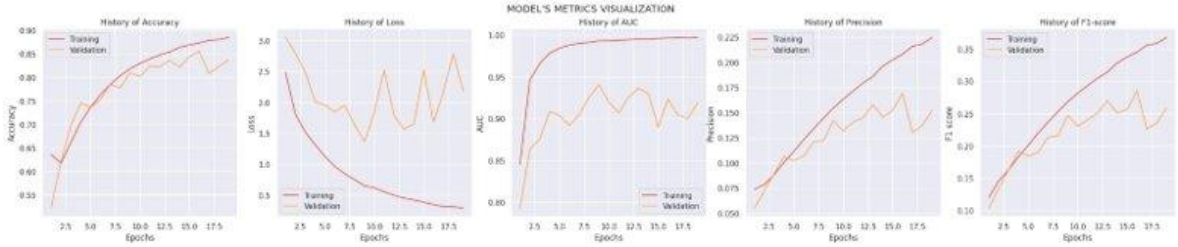
- Deep neural architecture with 48 layers
- Inception v3 uses batch normalization and dropout regularization to help prevent overfitting during training.
- Inception v3 is a convolutional neural network (CNN) architecture that uses a combination of 1x1, 3x3, and 5x5 convolutions, as well as pooling layers, to extract features from input images.

- **Xception**
  - These convolutions are composed of two separate stages: a Depthwise Convolution that applies a single filter to each input channel, followed by a Point-wise Convolution that applies a 1x1 filter to the output of the depthwise Convolution.
  - Xception uses Batch Normalization, which helps to stabilize the network during training and can speed up convergence.
  - The architecture also includes pooling layers and fully connected layers for classification. Overall, Xception has a total of about 22 million parameters.
- **VGG19**
  - It consists of 19 convolutional layers and 3 fully connected layers, making a total of 22 layers.
  - The architecture is based on a series of convolutional layers with small 3x3 filters, followed by max-pooling layers. This simple structure allows for a deep and powerful feature extraction process.
  - VGG - 16 uses batch normalization, which helps to stabilize the network during training and can speed up convergence.
  - The model has a large number of parameters, with around 143 million trainable parameters.
- **EfficientNetV2L**
  - The model has a large number of parameters, with around 143 million trainable parameters.
  - It includes a pointwise convolutional layer that is used to transform the channel dimensions of the input tensor.
  - The pointwise convolutional layer is a 1x1 convolution that applies a linear transformation to the channels of the input tensor.
  - The architecture of EfficientNetV2L is based on a combination of several key techniques, including depth-wise separable convolutions, pointwise convolutions, and squeeze-and-excitation (SE) blocks.
  - The number of parameters in an EfficientNetV2L model depends on the specific variant of the model and the scale factor used.
- **NASNetLarge**
  - NASNetLarge is a neural network architecture that was designed using neural architecture search (NAS) to achieve state-of-the-art performance on image classification tasks.
  - The architecture of NASNetLarge consists of 1056 layers in total.

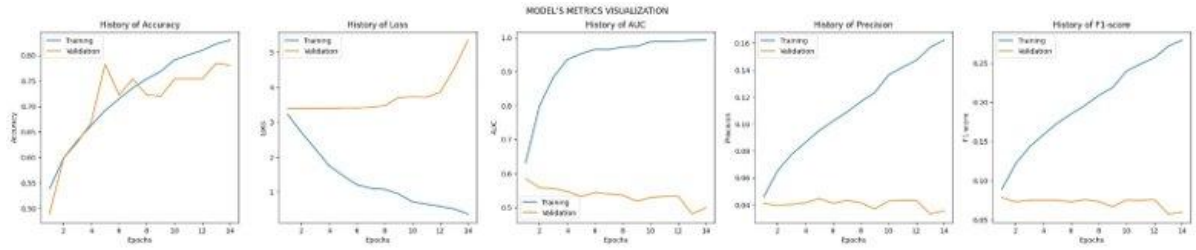
- NASNetLarge is a convolutional neural network architecture that was designed using neural architecture search which includes a series of blocks, consisting of normal cells and reduction cells, that are connected in a hierarchical fashion.
- NASNetLarge has approximately 88 million parameters, which is relatively small compared to other state-of-the-art models.

Evaluation of Original Data

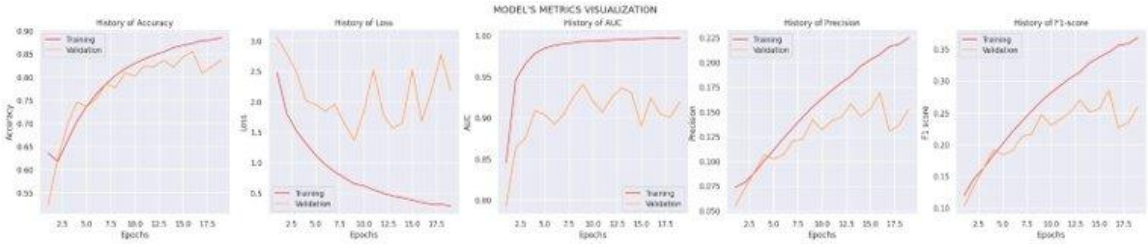
- Training and Validation Plots for Original Data.
  - VGG19



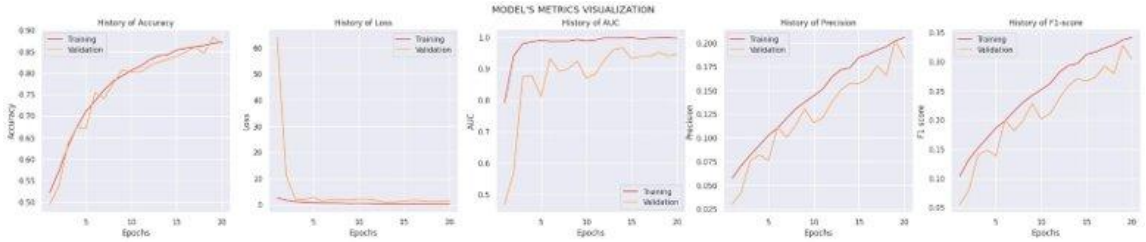
  - NASNetLarge



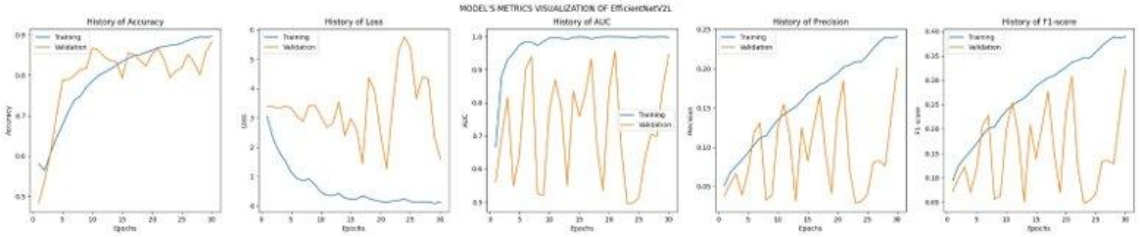
  - ResNet50



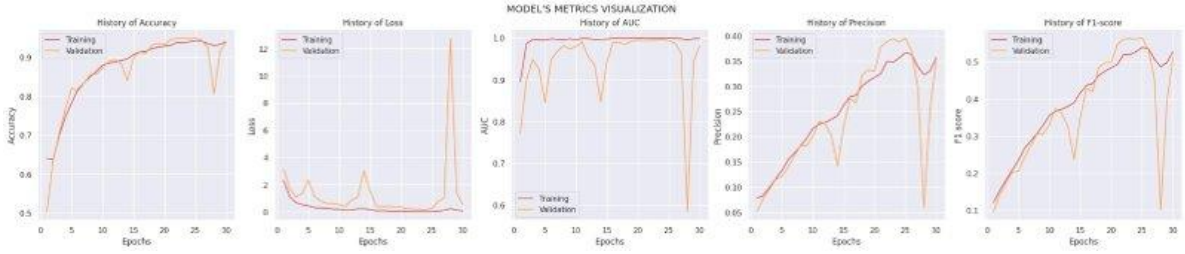
  - InceptionV3



- EfficientNetV2L

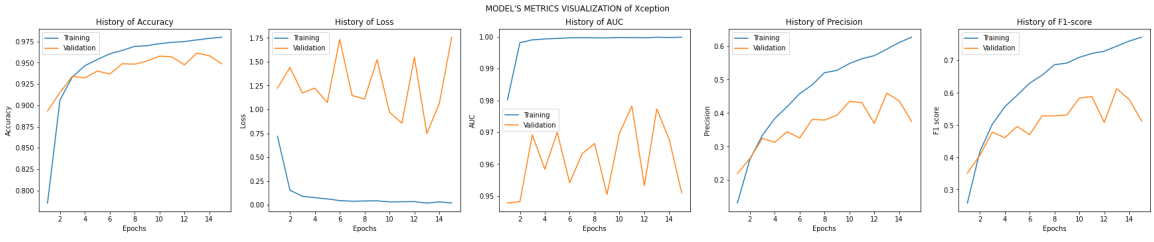


- Xception



- Xception (Augmented~8K Samples)

- Xception (Augmented~30K Samples)



- Results on Original Data on Test Set

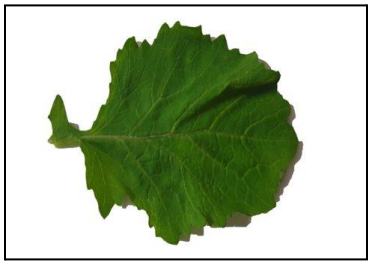
Model	Accuracy	Precision	Recall	AUC	F-1 Score
VGG19	0.6226	0.0707	0.850	0.8089	0.1308
NASNetLarge	0.7808	0.0351	0.2105	0.4978	0.0601
ResNet50	0.8410	0.1612	0.8974	0.9354	0.2753
InceptionV3	0.8711	0.1908	0.8842	0.9536	0.3144
EfficientNetV2L	0.8825	0.2030	0.8632	0.9475	0.3292
Xception	**0.9418**	**0.3569**	**0.9289**	**0.9763**	**0.5173**
Xception (Augmented~ 8K Samples) + 30 Epochs	**0.9575**	**0.4385**	**0.9804**	**0.9960**	**0.6083**
Xception(Augmented~ 30K Samples) + 15 Epochs	**0.9723**	**0.5486**	**0.9507**	**0.9893**	**0.6984**

● Explainability using GRAD-CAM-

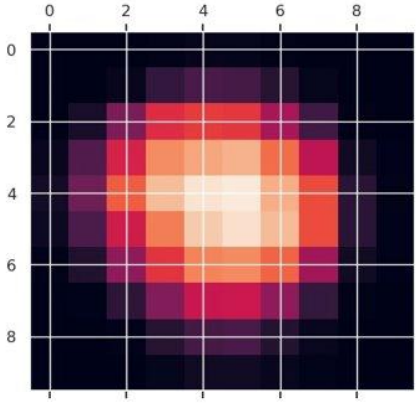
○ Definition

- Explainability refers to the ability to understand and interpret the decision-making processes of an algorithm or model.
- Explainability can help researchers understand the factors that influence their results, and can provide insights into the underlying mechanisms behind their findings.
- Explainability can enhance trust in the research findings, particularly when the results have important implications for society. By making the decision-making processes of a model transparent, researchers can help to alleviate concerns about bias or unfairness.
- GRAD-CAM as a TOOL for explainability - GRAD-CAM (Gradient-weighted Class Activation Mapping) is a technique used for visualizing and interpreting the decisions made by deep neural networks in image classification tasks.

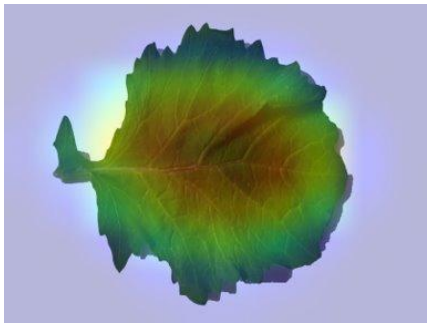
- **Original Image -**



- **HEATMAP - For GRADIENT HEATMAP, GRAD-CAM** works by computing the gradient of the output class score with respect to the feature maps of the last convolutional layer in the neural network. These gradients represent how much each feature map contributes to the classification decision for a given input image. The gradients are then used to weight the feature maps, producing a heatmap that highlights the regions of the input image that were most important for the classification decision.



- **FINAL RESULT IMAGE** - The resulting heatmap can be overlaid on the original input image, allowing researchers to visually interpret which regions of the image were most influential in the classification decision. GRAD-CAM is particularly useful in cases where the neural network is making decisions based on features that are not easily identifiable to humans, such as in medical imaging or object detection tasks.

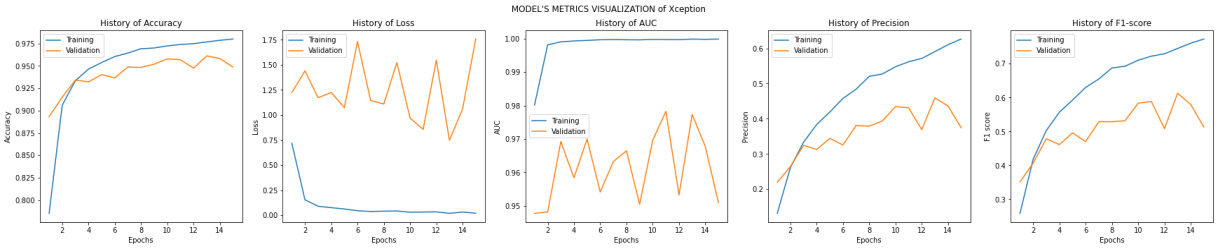


- The best model trained is ***Xception*** and hence ***Xception*** is further trained on an enhanced dataset of 30,000 Samples which consists of several different types of augmentations on the given dataset.
- The details of the number of samples in the Training, Validation and Test Set is as follows : -
    - **For Train Set - 19967 images belonging to 30 Classes**
    - **For Validation Set - 4977 images belonging to 30 Classes.**
    - **For Test Set - 6251 images belonging to 30 Classes.**
- The model architecture is as follows:-
- The Training and Validation History of the model for ***15 Epochs*** is as follows :-

```
Epoch 5/15
624/624 [==============================] - 785s 1s/step - loss: 0.0625 - accuracy: 0.9538 - precision: 0.4192 - recall: 0.9981
- auc: 0.9995 - get_f1_score: 0.5927 - val_loss: 1.0731 - val_accuracy: 0.9403 - val_precision: 0.3443 - val_recall: 0.8754 - v
al_auc: 0.9700 - val_get_f1_score: 0.4958
Epoch 6/15
624/624 [==============================] - 785s 1s/step - loss: 0.0449 - accuracy: 0.9605 - precision: 0.4577 - recall: 0.9988
- auc: 0.9997 - get_f1_score: 0.6300 - val_loss: 1.7331 - val_accuracy: 0.9367 - val_precision: 0.3256 - val_recall: 0.8387 - v
al_auc: 0.9541 - val_get_f1_score: 0.4702
Epoch 7/15
624/624 [==============================] - 788s 1s/step - loss: 0.0380 - accuracy: 0.9644 - precision: 0.4839 - recall: 0.9987
- auc: 0.9997 - get_f1_score: 0.6547 - val_loss: 1.1449 - val_accuracy: 0.9488 - val_precision: 0.3808 - val_recall: 0.8545 - v
al_auc: 0.9632 - val_get_f1_score: 0.5287
Epoch 8/15
624/624 [==============================] - 786s 1s/step - loss: 0.0409 - accuracy: 0.9693 - precision: 0.5205 - recall: 0.9987
- auc: 0.9997 - get_f1_score: 0.6873 - val_loss: 1.1098 - val_accuracy: 0.9484 - val_precision: 0.3791 - val_recall: 0.8612 - v
al_auc: 0.9665 - val_get_f1_score: 0.5286
Epoch 9/15
624/624 [==============================] - 788s 1s/step - loss: 0.0427 - accuracy: 0.9701 - precision: 0.5273 - recall: 0.9982
- auc: 0.9996 - get_f1_score: 0.6925 - val_loss: 1.5223 - val_accuracy: 0.9520 - val_precision: 0.3936 - val_recall: 0.8123 - v
al_auc: 0.9504 - val_get_f1_score: 0.5317
Epoch 10/15
624/624 [==============================] - 786s 1s/step - loss: 0.0307 - accuracy: 0.9725 - precision: 0.5477 - recall: 0.9987
- auc: 0.9998 - get_f1_score: 0.7102 - val_loss: 0.9702 - val_accuracy: 0.9579 - val_precision: 0.4346 - val_recall: 0.8774 - v
al_auc: 0.9696 - val_get_f1_score: 0.5835
Epoch 11/15
624/624 [==============================] - 786s 1s/step - loss: 0.0322 - accuracy: 0.9740 - precision: 0.5619 - recall: 0.9986
- auc: 0.9998 - get_f1_score: 0.7222 - val_loss: 0.8564 - val_accuracy: 0.9569 - val_precision: 0.4312 - val_recall: 0.9160 - v
al_auc: 0.9782 - val_get_f1_score: 0.5886
Epoch 12/15
624/624 [==============================] - 789s 1s/step - loss: 0.0335 - accuracy: 0.9750 - precision: 0.5714 - recall: 0.9983
- auc: 0.9997 - get_f1_score: 0.7292 - val_loss: 1.5476 - val_accuracy: 0.9476 - val_precision: 0.3691 - val_recall: 0.8087 - v
al_auc: 0.9532 - val_get_f1_score: 0.5084
Epoch 13/15
624/624 [==============================] - 791s 1s/step - loss: 0.0190 - accuracy: 0.9769 - precision: 0.5906 - recall: 0.9994
- auc: 0.9999 - get_f1_score: 0.7449 - val_loss: 0.7474 - val_accuracy: 0.9613 - val_precision: 0.4594 - val_recall: 0.9090 - v
al_auc: 0.9773 - val_get_f1_score: 0.6128
Epoch 14/15
624/624 [==============================] - 788s 1s/step - loss: 0.0316 - accuracy: 0.9787 - precision: 0.6104 - recall: 0.9988
- auc: 0.9998 - get_f1_score: 0.7605 - val_loss: 1.0605 - val_accuracy: 0.9583 - val_precision: 0.4363 - val_recall: 0.8573 - v
al_auc: 0.9677 - val_get_f1_score: 0.5798
Epoch 15/15
624/624 [==============================] - 781s 1s/step - loss: 0.0203 - accuracy: 0.9801 - precision: 0.6264 - recall: 0.9994
- auc: 0.9999 - get_f1_score: 0.7727 - val_loss: 1.7574 - val_accuracy: 0.9488 - val_precision: 0.3747 - val_recall: 0.8021 - v
al_auc: 0.9510 - val_get_f1_score: 0.5130
```

RESULTS

A. Training and Validation Loss for Training and Validation Data respectively.

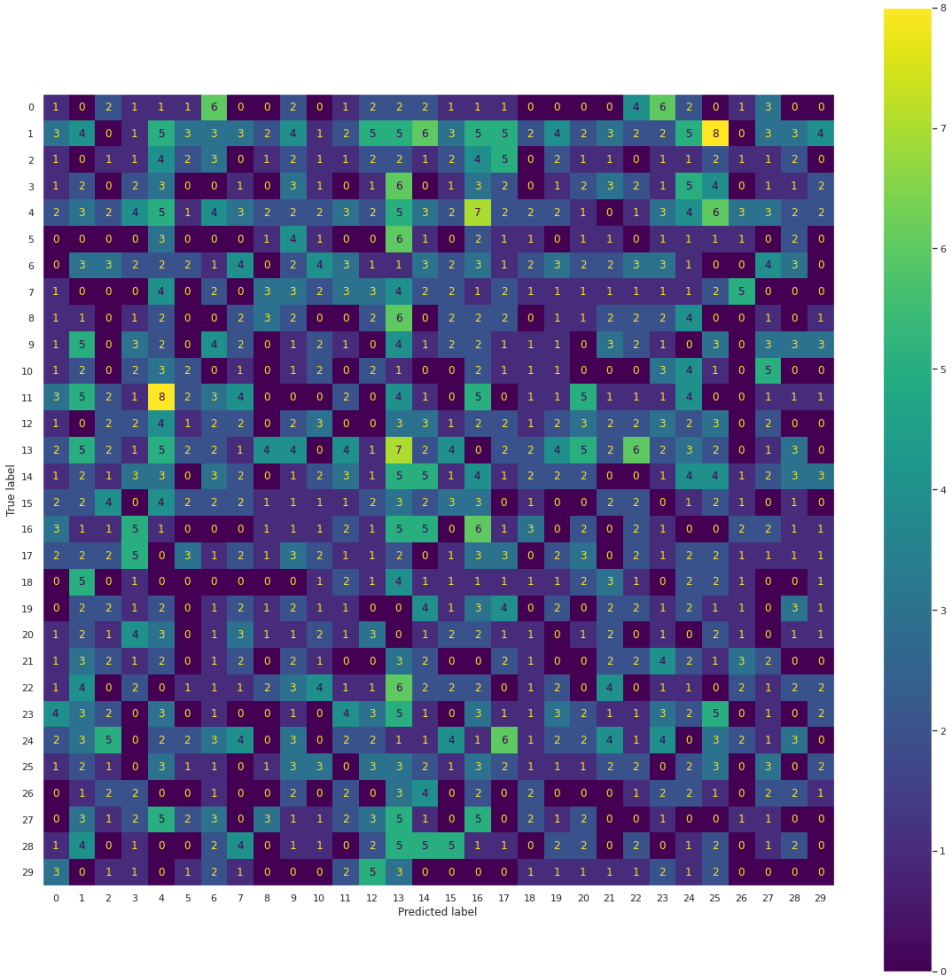


B. Loss, Accuracy, Precision, Recall and F-1 Score for Test Data

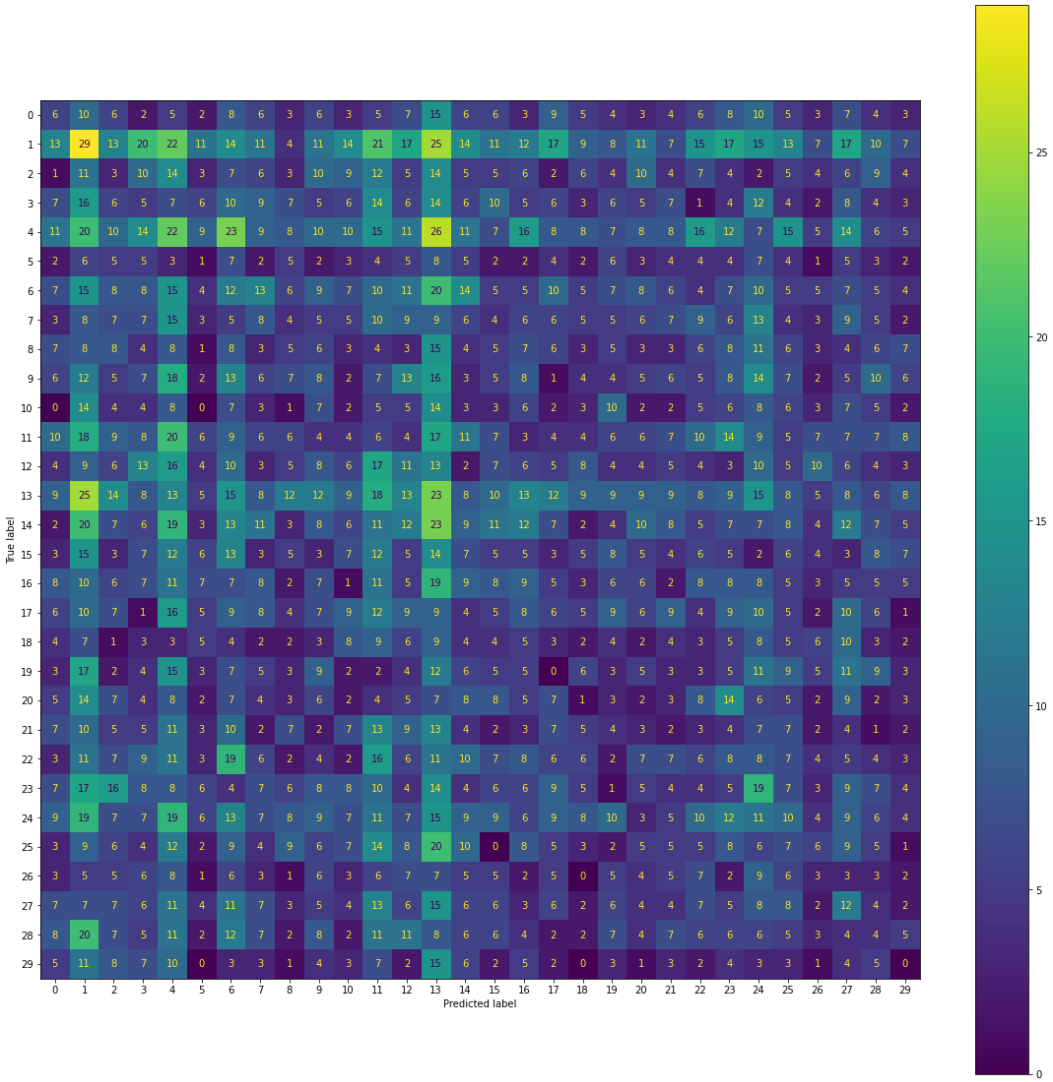
```
In [63]: res1 = get_test_metric(Xception_load)

196/196 - 201s - loss: 0.4496 - accuracy: 0.9723 - precision: 0.5486 - recall: 0.9507 - auc: 0.9893 - get_f1_score: 0.6984
```

C. The confusion matrix for the predicted labels and the true labels is as shown below -

a. Confusion Matrix (For Xception (~8K Samples)

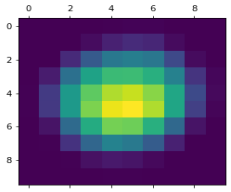
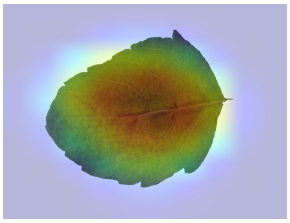
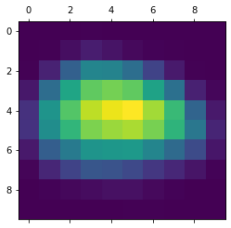
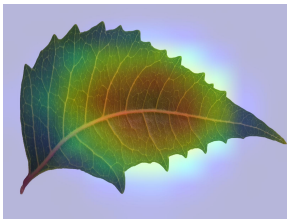
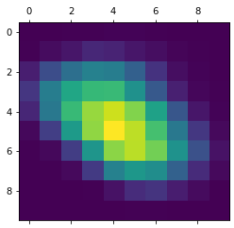
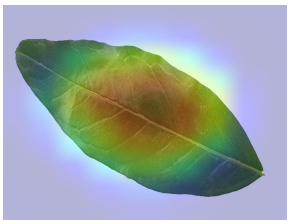
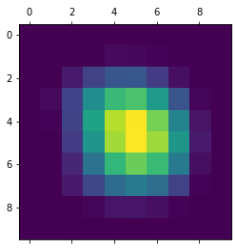
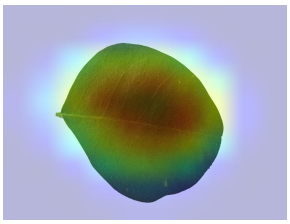


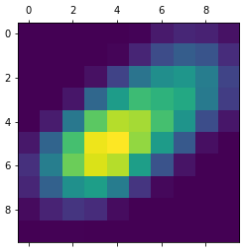
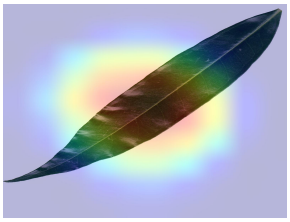
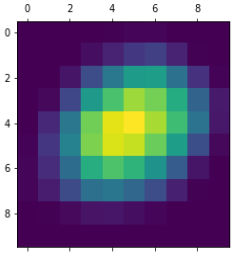
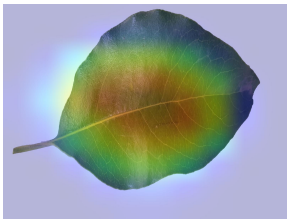
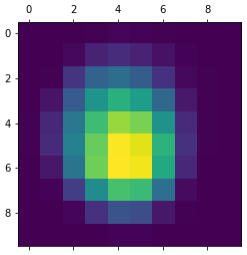
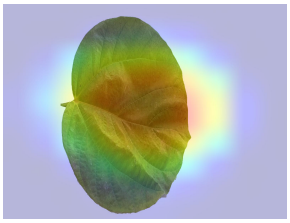
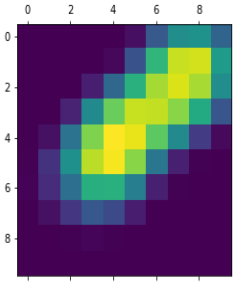
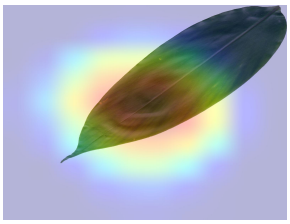
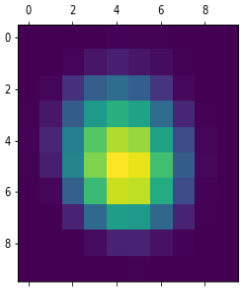
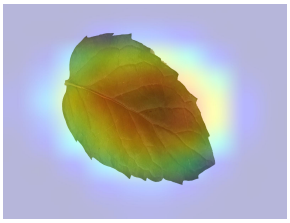
b. <u>Confusion Matrix (For Xception (~30K Samples)</u>

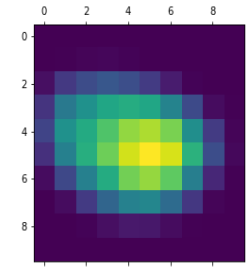
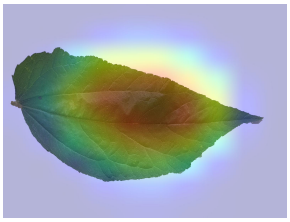
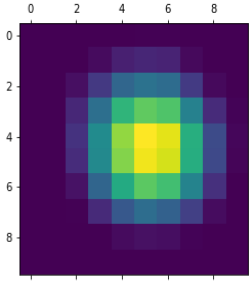
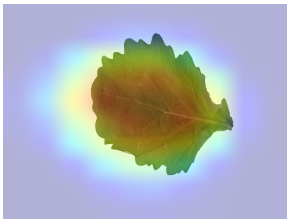
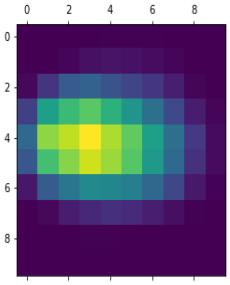
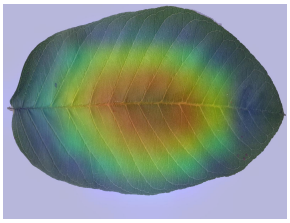
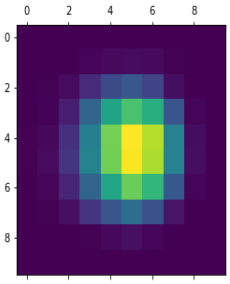
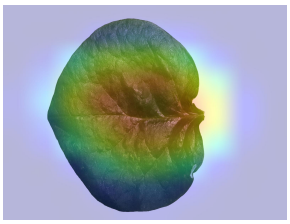
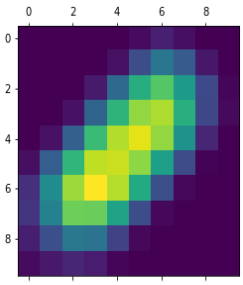
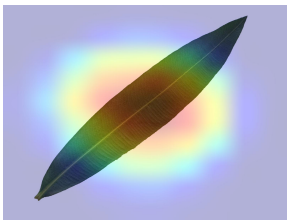


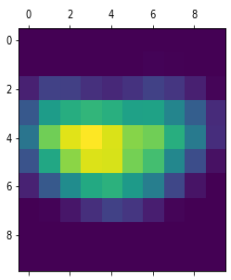
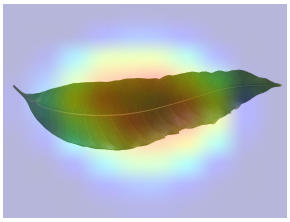
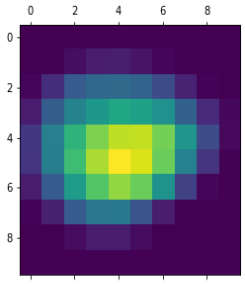
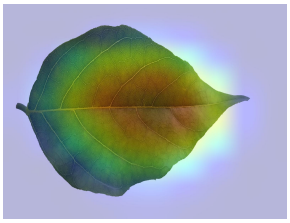
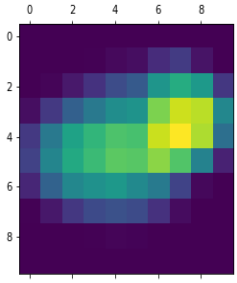
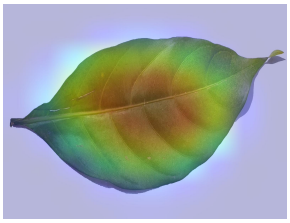
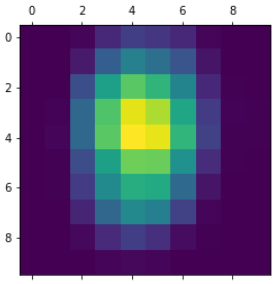
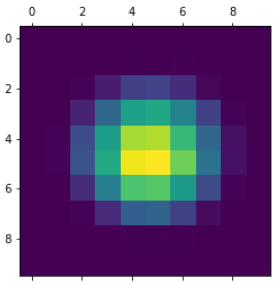
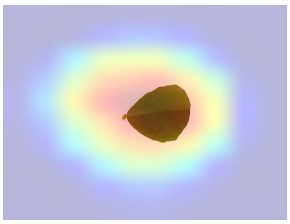
Explainability

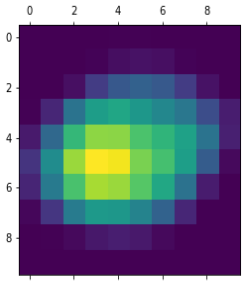
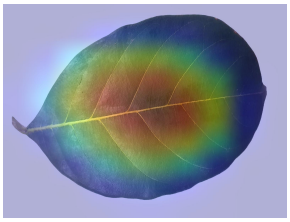
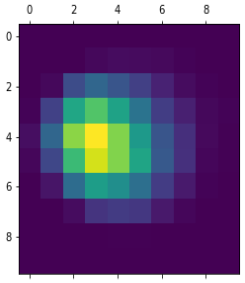
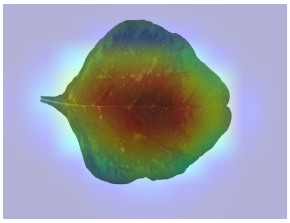
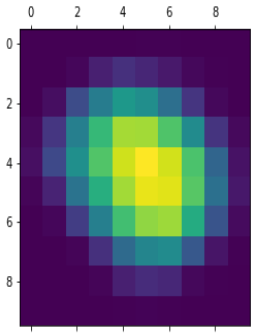
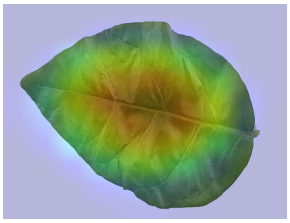
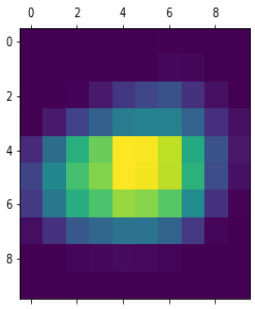
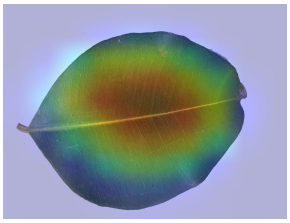
(For 30 Classes of the Dataset)

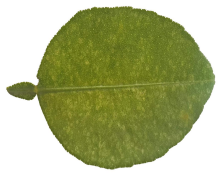
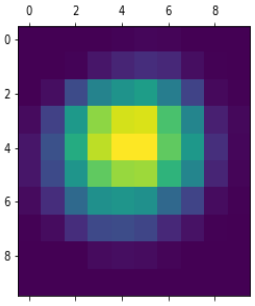
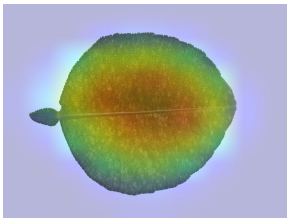
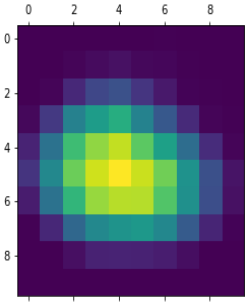
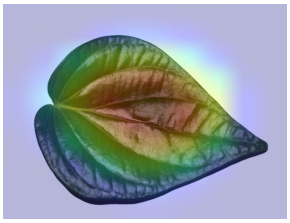
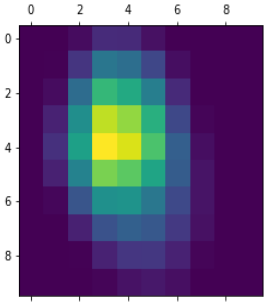
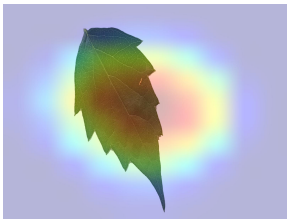
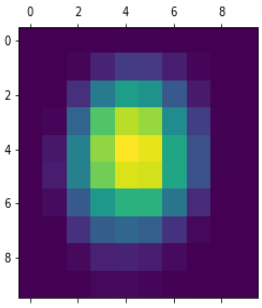
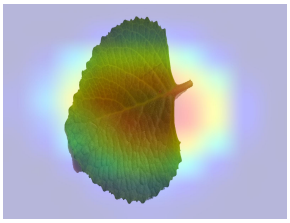
S.No.	Class	Original Image	Heatmap	GRAD-CAM
1.	Ocimum Tenuiflorum (Tulsi)			
2.	Azadirachta Indica (Neem)			
3.	Punica Granatum (Pomegranate)			
4	Punica Granatum (Pomegranate)			

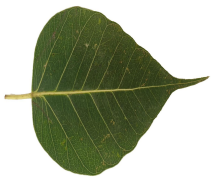
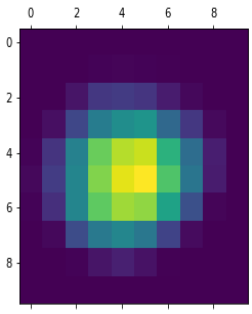
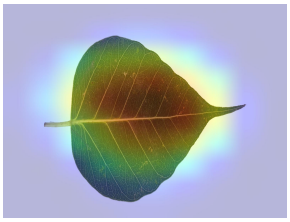
5	Moringa Oleifera (Drumstick)			
6	Syzygium Jambos (Rose Apple)			
7	Santalum Album (Sandalwood)			
8	Ficus Auriculata (Roxburgh)			
9	Mentha (Mint)			

10	Muntingia Calabura (Jamaica Cherry-Gasagase)			
11	Brassica Juncea (Indian Mustard)			
12	Psidium Guajava (Guava)			
13	Basella Alba (Basale)			
14	Nerium Oleander (Oleander)			

15	Mangifera Indica (Mango)			
16	Pongamia Pinnata (Indian Beech)			
17	Tabernaemontana Divaricata (Crape Jasmine)			
18	Hibiscus Rosa-sinensis			
19	Trigonella Foenum-graecum (Fenugreek)			

20	Artocarpus Heterophyllus (Jackfruit)			
21	Amaranthus Viridis (Arive-Dantu)			
22	Jasminum (Jasmine)			
23	Syzygium Cumini (Jamun)			

24	Citrus Limon (Lemon)			
25	Piper Betle (Betel)			
26	Nyctanthes Arbor-tristis (Parijata)			
27	Plectranthus Amboinicus (Mexican Mint)			

29	Ficus Religiosa (Peepal Tree)			
30	Murraya Koenigii (Curry)