DEEP LEARNING CS6005

Mini Project on Computer vision with Transfer Learning application

Plant Disease Identification using Transfer Learning

Date : 11-05-2021

Praveen RS

2018103577

P Batch

Problem Statement

Apples are one of the most important temperate fruit crops in the world. Foliar (leaf) diseases pose a major threat to the overall productivity and quality of apple orchards. The current process for disease diagnosis in apple orchards is based on manual scouting by humans, which is time-consuming and expensive.

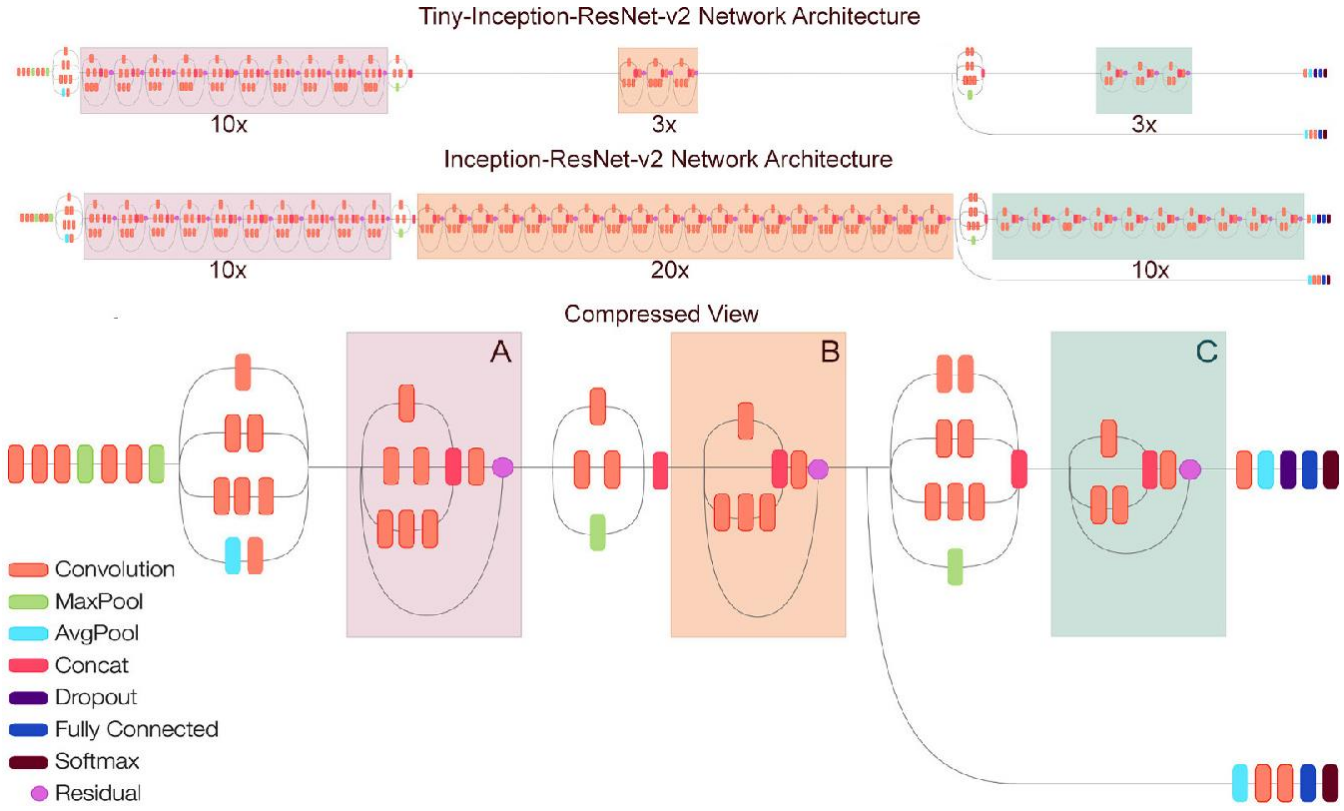
Hence, we use transfer learning to identify if the apple leaves are healthy or else have diseases like scab, complex, rust, frog eye leaf spot, powdery mildew and other leaf diseases.

Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

The intuition behind transfer learning for image classification is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world. You can then take advantage of these learned feature maps without having to start from scratch by training a large model on a large dataset.

In computer vision, for example, neural networks usually try to detect edges in the earlier layers, shapes in the middle layer and some task-specific features in the later layers. In transfer learning, the early and middle layers are used and we only retrain the latter layers. It helps leverage the labelled data of the task it was initially trained on.



We use **Inception-ResNet-v2** as the pre-trained model in this project. Inception-ResNet-v2 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 164 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

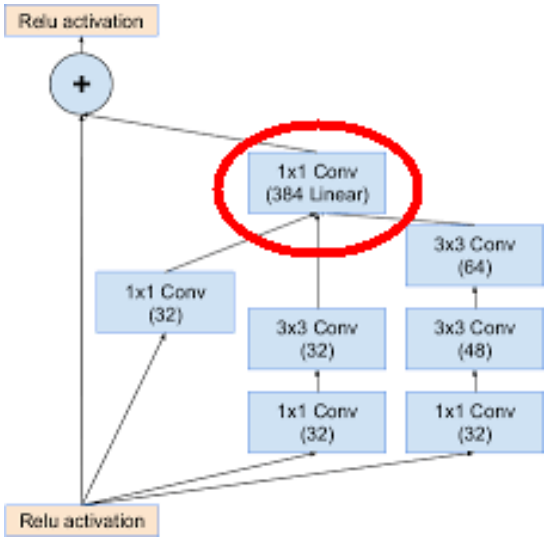
Both the Inception and Residual networks are SOTA architectures, which have shown very good performance with relatively low computational cost. Inception-ResNet combines the two architectures to further boost the performance.

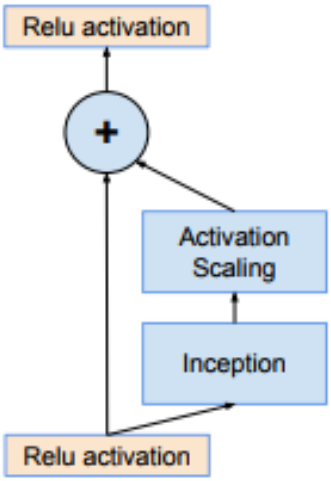
Architecture of the Inception-ResNet-v2 model:

1. Each Inception block is followed by a filter expansion layer (1 × 1 convolution without activation) which is used for scaling up the dimensionality of the filter bank before the addition to match the depth of the input.
2. In the case of Inception-ResNet, batch-normalization is used only on top of the traditional layers, but not on top of the summations.

If the number of filters exceeded 1000, the residual variants started to exhibit instabilities and the network has just “died” early in the training, meaning that the last layer before the average pooling started to produce only zeros after a few tens of thousands of iterations. This could not be prevented, neither by lowering the learning rate, nor by adding an extra batch-normalization to this layer.

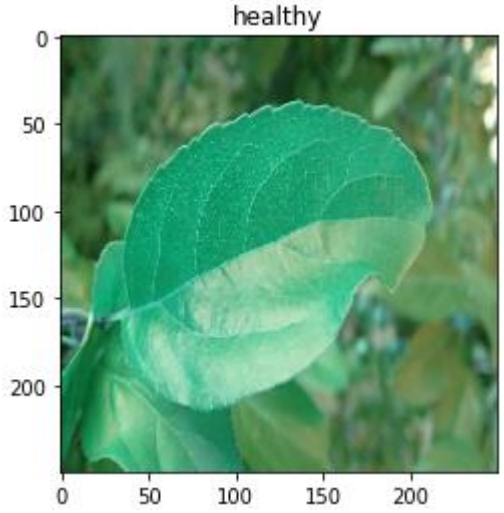
Scaling down the residuals before adding them to the previous layer activation seemed to stabilize the training. To scale the residuals, scaling factors between 0.1 and 0.3 are picked.

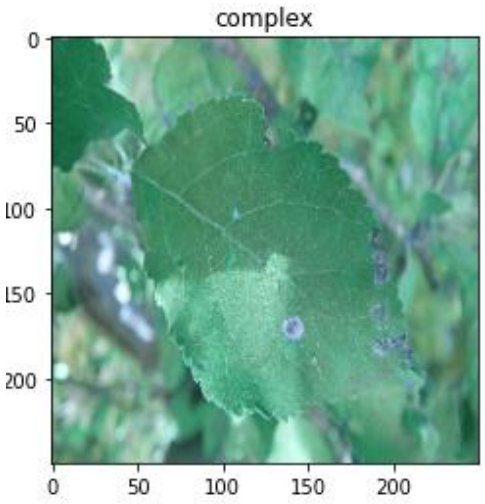


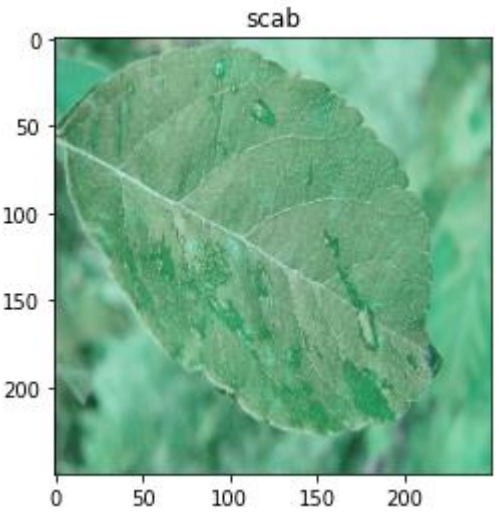


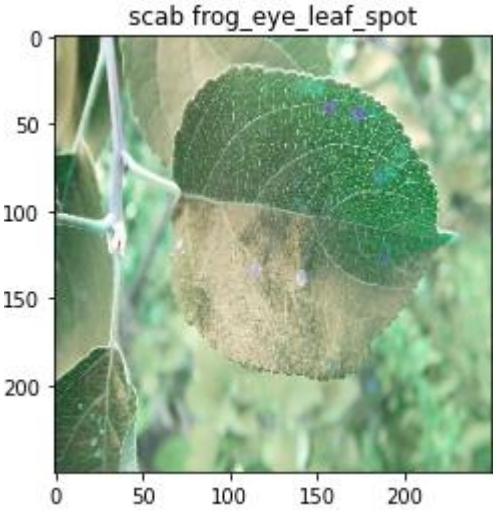
Dataset Details

The dataset used is taken from Plant Pathology 2021 challenge competition which had a pilot dataset of approximately 23,000 high-quality RGB images (in jpg format) of apple foliar diseases, including a large expert-annotated disease dataset. This dataset reflects real field scenarios by representing non-homogeneous backgrounds of leaf images taken at different maturity stages and at different times of day under different focal camera settings.









In this project, we will be using around 17,600 images as the training dataset and 1000 images for testing the final model.

Modules Used

1.Loading the dataset and necessary libraries

Firstly, the image folders and the corresponding '.csv' files are loaded into the program. Then, all the necessary libraries are imported for the execution of the program and visualization of the results. Now, we scale the dataset.

2. Creating Image data generator

Three different data generators are created for train, validation and test data. Image data generator is created with rescaling factor as 1/255 in all three. Feature-wise centre is set to true transform the images to 0. The validation data split is 10% of the total data available for training. The target size is set as 256, hence the images will be of the size (256,256) and color_mode is set to 'rgb', where 'rgb' denotes the RGB channels in the image. The batch size will be the default value 32.

3.Import Pre-trained Model

We will be using InceptionResNetV2 pre-trained model. The weights for this model are obtained from the Keras library and loaded into the program. Also, the 'include_top' parameter is set to 'False' to remove all the top layers for customization. For better prediction results, we will be freezing almost all of the InceptionResNetV2 layers except for a few hundred layers at the bottom.

In addition to this, we will be adding a GlobalAveragePooling2D layer and one last Dense layer with 6 nodes, one for each class with 'sigmoid' as activation, one node for each label(this is a multilabel classification problem) .

The loss function used for this final model is 'Binary Cross-entropy' and the optimizer is 'Adam'. The summary of the Final Model is as follows:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_resnet_v2 (Functio (None, 6, 6, 1536)        54336736
_________________________________________________________________
global_average_pooling2d (Gl (None, 1536)              0
_________________________________________________________________
dense (Dense)                (None, 6)                 9222
=================================================================
Total params: 54,345,958
Trainable params: 22,450,598
Non-trainable params: 31,895,360
_________________________________________________________________
```

4.Training the Model

The following parameters are used for training the model:

Optimizer	Adam
Loss	Binary Cross-entropy
Epochs	100
Batch size	32
Learning rate	0.01
Monitor	Validation loss

Since it is a multilabel image classification, we will be going for **F1 accuracy** instead of binary accuracy in macro mode. A macro-average will compute the metric independently for each class and then take the average (hence treating all classes equally).

We will also be using early-stopping and model checkpoint to avoid overfitting.

5.Predicting the Test data labels

The predicted values on the test data are obtained and the model is also evaluated for the f1 score and the loss on test set.

6.Visualizing the Results

The model's f1 score vs epochs graph and loss vs epochs graph are plotted for both training data and validation data. Since we don't get a categorical value from the result, we index the six classes as numerical labels from 0 to 5. Classification report is generated for the test data which contains information about precision, recall and f1-score, and the confusion matrix is plotted with the heatmap for visualization.

The classification report can't handle multilabel classifications hence we aggregate the indices from 2 to 6(complex, rust, frog eye leaf spot, powdery mildew) as simply 2.

Source Code

Importing the necessary libraries and datasets

```
[1]
import pandas as pd
import numpy as np
import tensorflow as tf
import PIL
import cv2
from keras_preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
import tensorflow_addons as tfadd
import seaborn as sns
import matplotlib.pyplot as plt
from tensorflow.keras.applications import InceptionResNetV2
```

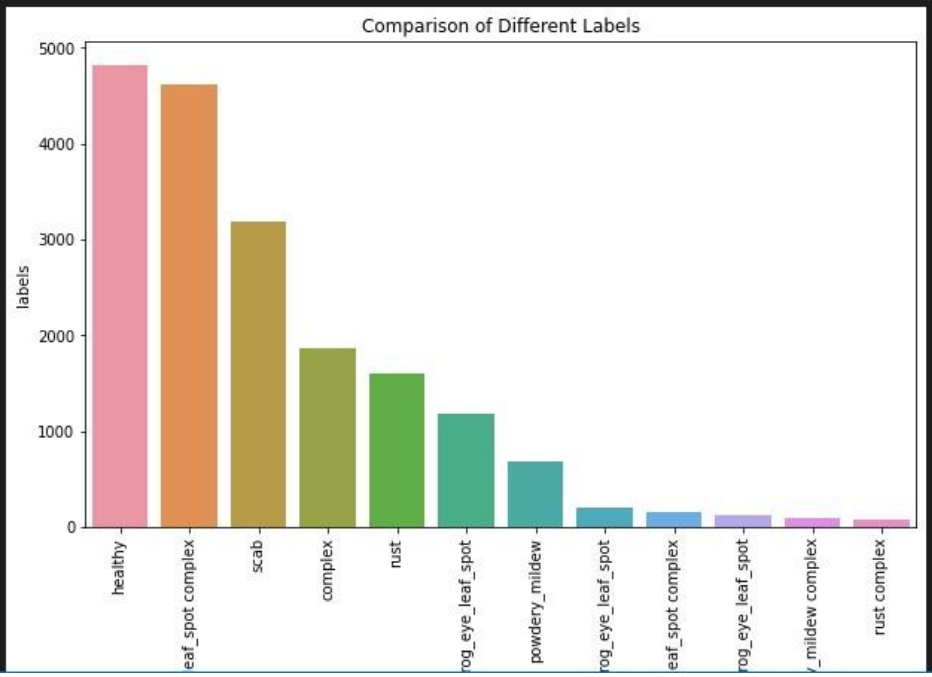
Now, we will set the data directories for images and datasets

```
[2]
train_data = pd.read_csv('D:/Sem 6/Deep Learning/Assigments/TL_Mini_Project/train.csv')
test_data = pd.read_csv('D:/Sem 6/Deep Learning/Assigments/TL_Mini_Project/test.csv')
TRAIN_IMG_DIR = 'D:/Sem 6/Deep Learning/Assigments/TL_Mini_Project/Train/'
TEST_IMG_DIR = 'D:/Sem 6/Deep Learning/Assigments/TL_Mini_Project/Test/'
```

Comparison of Different labels

```
plt.figure(figsize=(10, 6))
plt.xticks(rotation=90)
plt.title("Comparison of Different Labels")
sns.barplot(x=labels, y=value_counts)
```

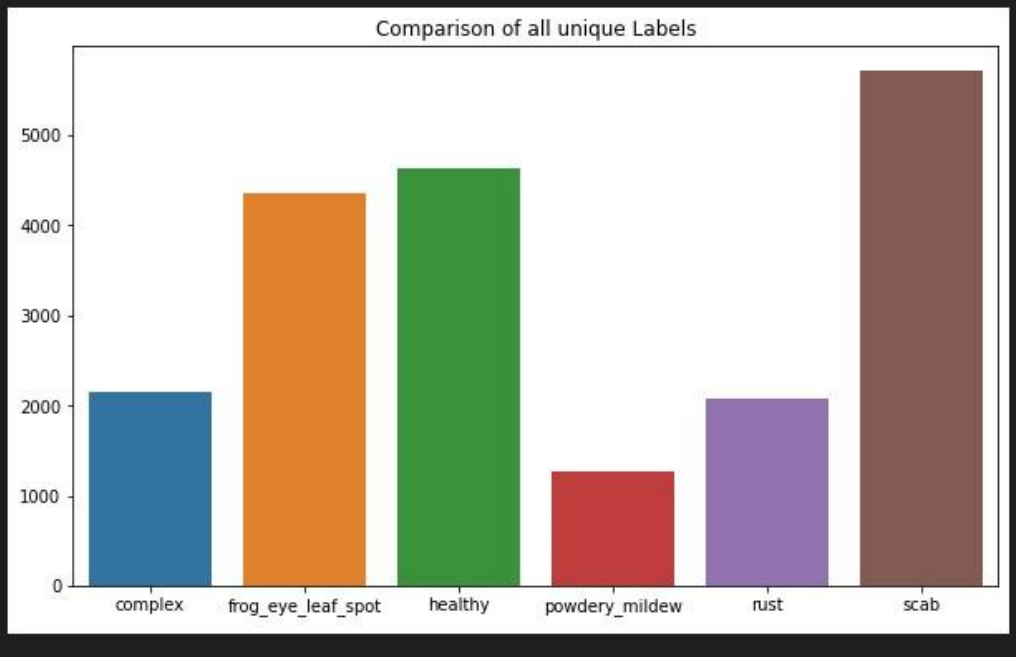
```
<AxesSubplot:title={'center':'Comparison of Different Labels'}, ylabel='labels'>
```



Comparison of Unique labels

```
plt.figure(figsize=(10, 6))
plt.title("Comparison of all unique Labels")
sns.barplot(x=cols, y=label_counts)
```

```
<AxesSubplot:title={'center':'Comparison of all unique Labels'}>
```



Creating Image Data Generator for Train data and Test data

[10]

```
train_data['labels'] = train_data['labels'].str.split(" ")

datagen = ImageDataGenerator(rescale=1./255, validation_split=0.1)
final_train_data = datagen.flow_from_dataframe(train_data,
    directory='/kaggle/input/resized-plant2021/img_sz_512',
    x_col="image",
    y_col="labels",
    target_size=(256, 256),
    color_mode="rgb",
    class_mode="categorical",
    subset="training")
```

```
Found 16769 validated image filenames belonging to 6 classes.
```

[11]

```
validation_data = datagen.flow_from_dataframe(train_data,
    directory='/kaggle/input/resized-plant2021/img_sz_512',
    x_col="image",
    y_col="labels",
    target_size=(256, 256),
    color_mode="rgb",
    class_mode="categorical",
    subset="validation")
```

```
Found 1863 validated image filenames belonging to 6 classes.
```

```
[12]
test_data['labels'] = test_data['labels'].str.split(" ")

test_datagen = ImageDataGenerator(rescale=1./255)

final_test_data = test_datagen.flow_from_dataframe(test_data,
    directory='/kaggle/input/testdata/Test',
    x_col="image",
    y_col="labels",
    target_size=(256, 256),
    color_mode="rgb",
    class_mode="categorical")
```

```
Found 1000 validated image filenames belonging to 6 classes.
```

Importing Inception ResNetV2 model

```
[13]
weights = 'D:/Sem 6/Deep Learning/Assigments/TL_Mini_Project/
inception_resnet_v2_weights_tf_dim_ordering_tf_kernels_notop.h5'

pretrained_weight_model = InceptionResNetV2(
    include_top=False,
    weights=weights,
    input_shape=(256, 256, 3)
)
```

```
[14]
pretrained_weight_model.input
pretrained_weight_model.output
```

```
<KerasTensor: shape=(None, 6, 6, 1536) dtype=float32 (created by layer 'conv_7b_ac')>
```

Unfreezing few layers

```
[15]
len(pretrained_weight_model.layers)
```

```
780
```

```
[16]
pretrained_weight_model.input
pretrained_weight_model.output
```

```
<KerasTensor: shape=(None, 6, 6, 1536) dtype=float32 (created by layer 'conv_7b_ac')>
```

```
[17]
for layer in pretrained_weight_model.layers[:630]:
    layer.trainable = False
for layer in pretrained_weight_model.layers[630:]:
    layer.trainable = True
```

Final Model for Training

```
[18]
final_model = Sequential([
    pretrained_weight_model,
    GlobalAveragePooling2D(),
    Dense(units=6, activation = 'sigmoid')
])

final_model.summary()
```

Training starts

```
[19]
from tensorflow.keras.callbacks import ModelCheckpoint

f1_score = tfadd.metrics.F1Score(num_classes=6, average='macro')

early_stopping = EarlyStopping(monitor=f1_score, patience=3, mode='max', restore_best_weights=True)


final_model.compile(loss='binary_crossentropy', optimizer=Adam(epsilon=0.01),
              metrics= [f1_score])

final_model_save = ModelCheckpoint('weights.h5' ,
                                   save_best_only = True,
                                  save_weights_only=True,
                                  monitor='val_loss',mode='min',verbose=1)

history = final_model.fit(final_train_data, epochs=100,
        callbacks=early_stopping, validation_data=validation_data)
```

Model trains for 100 epochs

```
Epoch 1/100
525/525 [==============================] - 243s 433ms/step - loss: 0.3833 - f1_score: 0.3775 - val_loss: 0.2077 - val_f1_score: 0.7080
Epoch 2/100
525/525 [==============================] - 155s 295ms/step - loss: 0.1689 - f1_score: 0.7740 - val_loss: 0.1893 - val_f1_score: 0.7672
Epoch 3/100
525/525 [==============================] - 153s 291ms/step - loss: 0.1195 - f1_score: 0.8488 - val_loss: 0.1823 - val_f1_score: 0.7775
Epoch 4/100
525/525 [==============================] - 155s 295ms/step - loss: 0.0908 - f1_score: 0.8882 - val_loss: 0.1905 - val_f1_score: 0.7792
Epoch 5/100
525/525 [==============================] - 154s 294ms/step - loss: 0.0654 - f1_score: 0.9209 - val_loss: 0.2047 - val_f1_score: 0.7758
Epoch 6/100
525/525 [==============================] - 157s 298ms/step - loss: 0.0543 - f1_score: 0.9262 - val_loss: 0.2355 - val_f1_score: 0.7647
Epoch 7/100
525/525 [==============================] - 156s 298ms/step - loss: 0.0449 - f1_score: 0.9369 - val_loss: 0.2325 - val_f1_score: 0.7776
Epoch 8/100
525/525 [==============================] - 158s 301ms/step - loss: 0.0345 - f1_score: 0.9450 - val_loss: 0.2407 - val_f1_score: 0.7943
Epoch 9/100
525/525 [==============================] - 160s 304ms/step - loss: 0.0259 - f1_score: 0.9500 - val_loss: 0.2554 - val_f1_score: 0.7817
Epoch 10/100
525/525 [==============================] - 162s 308ms/step - loss: 0.0195 - f1_score: 0.9513 - val_loss: 0.2876 - val_f1_score: 0.7580
Epoch 11/100
525/525 [==============================] - 164s 313ms/step - loss: 0.0192 - f1_score: 0.9507 - val_loss: 0.2637 - val_f1_score: 0.7755
```

```
Epoch 90/100
525/525 [==============================] - 165s 314ms/step - loss: 0.0015 - f1_score: 0.9583 - val_loss: 0.3378 - val_f1_score: 0.7959
Epoch 91/100
525/525 [==============================] - 153s 290ms/step - loss: 8.2972e-04 - f1_score: 0.9605 - val_loss: 0.3395 - val_f1_score: 0.
8024
Epoch 92/100
525/525 [==============================] - 164s 313ms/step - loss: 0.0010 - f1_score: 0.9594 - val_loss: 0.3537 - val_f1_score: 0.7920
Epoch 93/100
525/525 [==============================] - 152s 289ms/step - loss: 0.0019 - f1_score: 0.9600 - val_loss: 0.3598 - val_f1_score: 0.7648
Epoch 94/100
525/525 [==============================] - 163s 311ms/step - loss: 0.0034 - f1_score: 0.9582 - val_loss: 0.3433 - val_f1_score: 0.7880
Epoch 95/100
525/525 [==============================] - 162s 309ms/step - loss: 0.0016 - f1_score: 0.9618 - val_loss: 0.3344 - val_f1_score: 0.8037
Epoch 96/100
525/525 [==============================] - 151s 287ms/step - loss: 0.0022 - f1_score: 0.9597 - val_loss: 0.3417 - val_f1_score: 0.8061
Epoch 97/100
525/525 [==============================] - 164s 312ms/step - loss: 0.0017 - f1_score: 0.9588 - val_loss: 0.3484 - val_f1_score: 0.8007
Epoch 98/100
525/525 [==============================] - 152s 290ms/step - loss: 0.0013 - f1_score: 0.9601 - val_loss: 0.6879 - val_f1_score: 0.7360
Epoch 99/100
525/525 [==============================] - 164s 312ms/step - loss: 0.0029 - f1_score: 0.9549 - val_loss: 0.3189 - val_f1_score: 0.8018
Epoch 100/100
525/525 [==============================] - 151s 288ms/step - loss: 0.0020 - f1_score: 0.9576 - val_loss: 0.3397 - val_f1_score: 0.8032
```

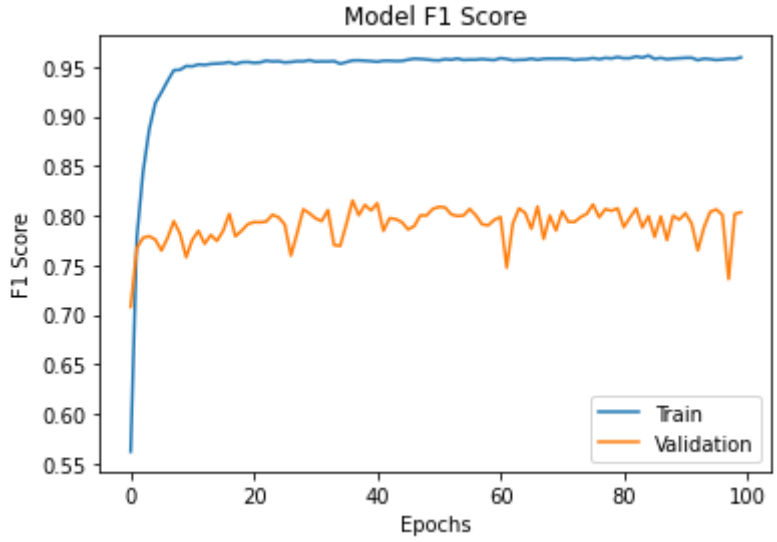
Train vs Validation results

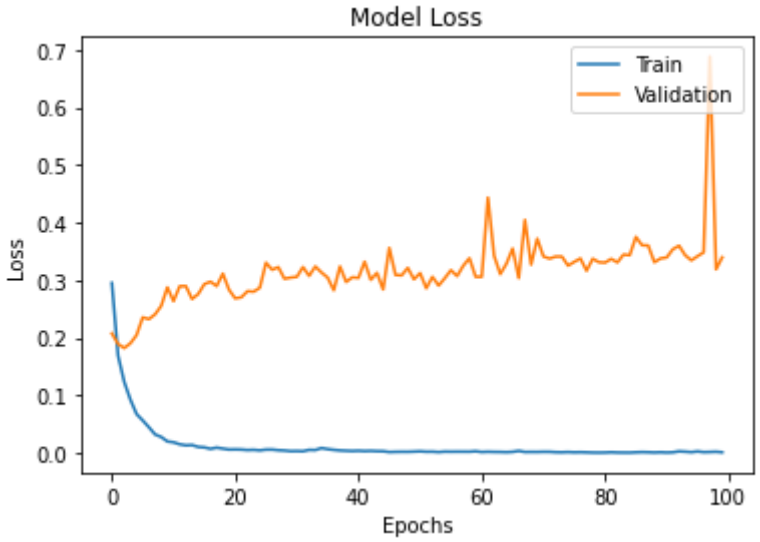
[20]
```
print(history.history.keys())
```
```
dict_keys(['loss', 'f1_score', 'val_loss', 'val_f1_score'])
```

[21]
```
# 'F1 Score'

plt.plot(history.history['f1_score'])
plt.plot(history.history['val_f1_score'])
plt.title('Model F1 Score')
plt.ylabel('F1 Score')
plt.xlabel('Epochs')
plt.legend(['Train','Validation'],loc='lower right')
plt.show()
```

[22]
```
# 'Loss'

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Train','Validation'],loc='upper right')
plt.show()
```

F1 Score and Loss Vs Epochs for Training and Validation





Prediction on Test Data

```
[25]  M↓
      pred_data = final_model.predict(final_test_data)
[26]  M↓
      pred_data
```

```
[33]  M↓
      final_model.evaluate(final_test_data)
32/32 [==============================] - 7s 224ms/step - loss: 3.0083e-04 - f1_score: 0.9670
[0.0003008329076692462, 0.9669587016105652]
```

Test data loss: 0.003

Test data F1 Score: 0.9669

Test Labels

```
[23]
final_test_data.classes

[[0],
 [3, 0],
 [1],
 [0],
 [2],
 [2],
 [2],
 [2],
 [3],
 [1],
 [1],
```

Predicted Labels

```
[29]
index_list

[[2],
 [2],
 [2],
 [2],
 [5],
 [2],
 [0],
 [5],
 [2],
 [2],
 [2],
 [0],
 [2],
 [1],
 [5],
```

```
[31]
pred_labels = final_train_data.class_indices
pred_labels = dict((value, key) for key, value in pred_labels.items())

pred_label_names = []

for indices in index_list:
    index = []
    for i in indices:
        index.append(str(pred_labels[i]))
    pred_label_names.append(' '.join(index))
```

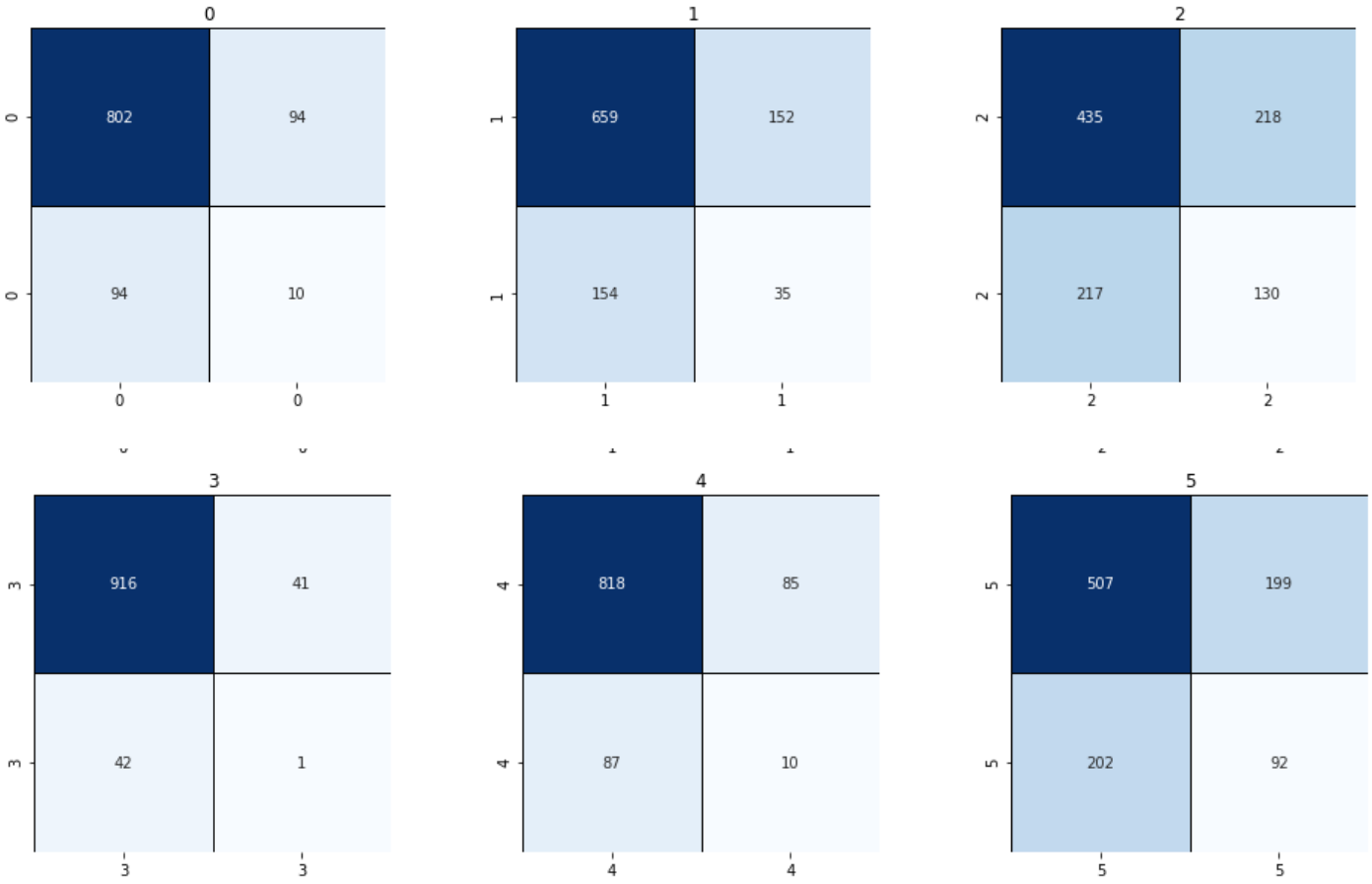
```
[32]
pred_label_names[0:10]

['healthy',
 'healthy',
 'healthy',
 'healthy',
 'scab',
 'healthy',
 'complex',
 'scab',
 'healthy',
 'healthy']
```

Confusion Matrix and Classification Report

```
Confusion Matrix:
[[[802  94]
  [ 94  10]]

 [[659 152]
  [154  35]]

 [[435 218]
  [217 130]]

 [[916  41]
  [ 42   1]]

 [[818  85]
  [ 87  10]]

 [[507 199]
  [202  92]]]
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	17
1	0.93	0.87	0.90	15
2	0.89	0.94	0.92	18
accuracy			0.94	50
macro avg	0.94	0.94	0.94	50
weighted avg	0.94	0.94	0.94	50

Heatmap for 6 Labels



Conclusion

We used the Inception ResNetV2 model to classify various leaf images into healthy or the disease which they have. We have obtained very good training and testing results and also in a comparatively short period of time due to transfer learning.