# Melanoma Detection

Melanoma Skin Cancer Detection using CNN

By: Himanshu Yadav, Amit Kumar Chaudhary & Shreeragh Chandrabose

# Problem Statement

- MELANOMA is a type of cancer that can be deadly if not detected early. It accounts for 75% of skin cancer deaths.
- A solution that can evaluate images and alert dermatologists about the presence of melanoma has the potential to reduce a lot of manual effort needed in diagnosis.
- Build a CNN based model which can accurately detect MELANOMA

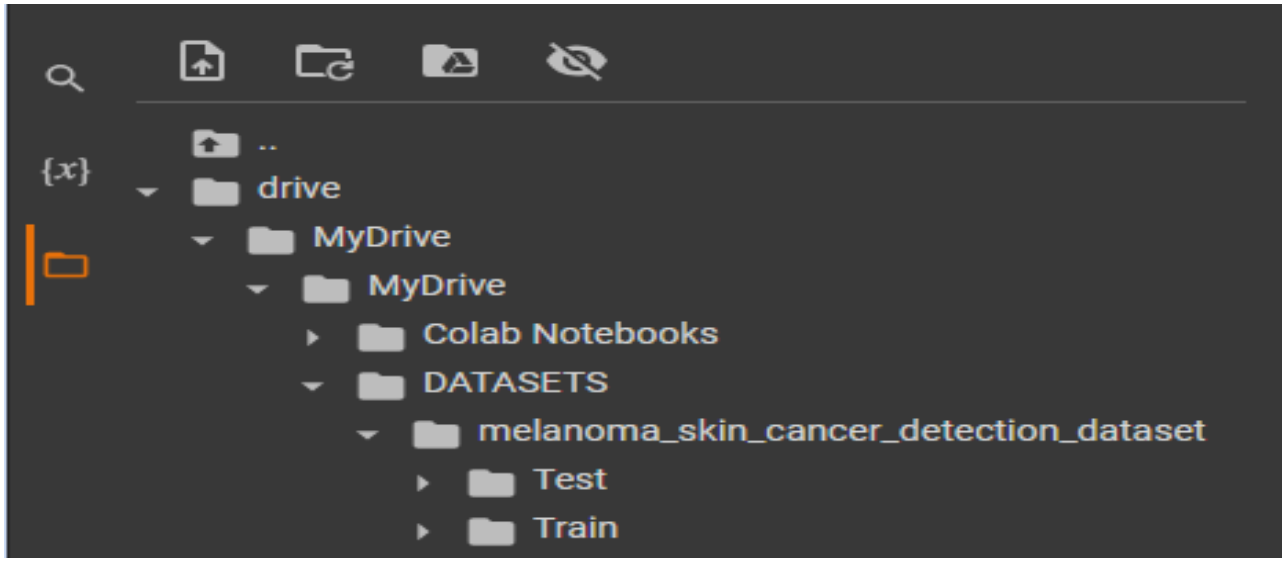
# Steps Taken:

- Load data to Google Drive and Mount on Colab
- Set train and test paths and split Train data into 80% for Training and 20% for Validation
- Set a primary baseline model and train it and check performance
- Try augmenting without ‘AUGMENTOR’ library to understand if we are headed to the right path using the same baseline optimized model
- As overfitting as found to be reducing, Using Augmentor Library to add 500 samples in each class and reduce Class Imbalance
- Check performance of baseline model on augmented dataset

# Mounting the Data

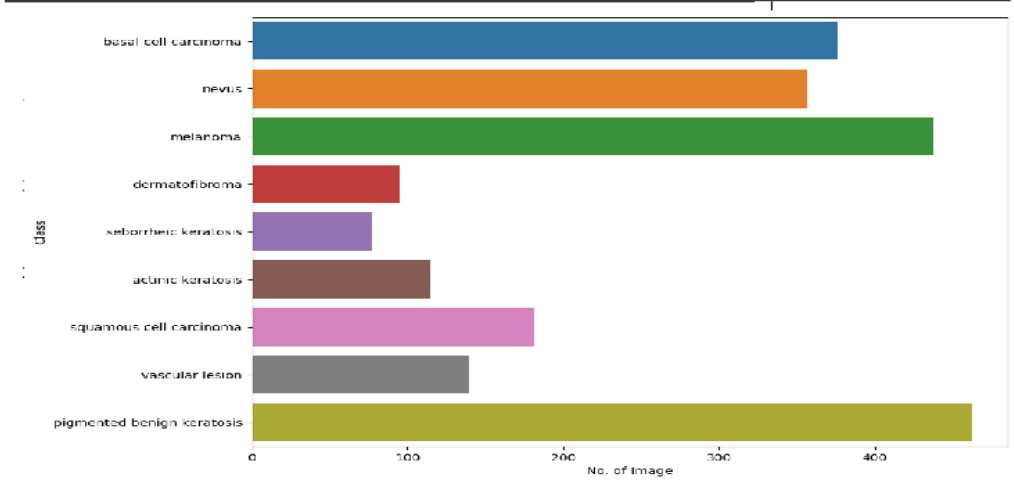
- Here is the snippet of code used to mount datasets in Colab and Drive Path Snippet
- Once mounted the image count was taken to check the count
- Also, it was ensured if the images were of size : 180 x 180
- Class names were printed to understand all the classes data needs to be classified into

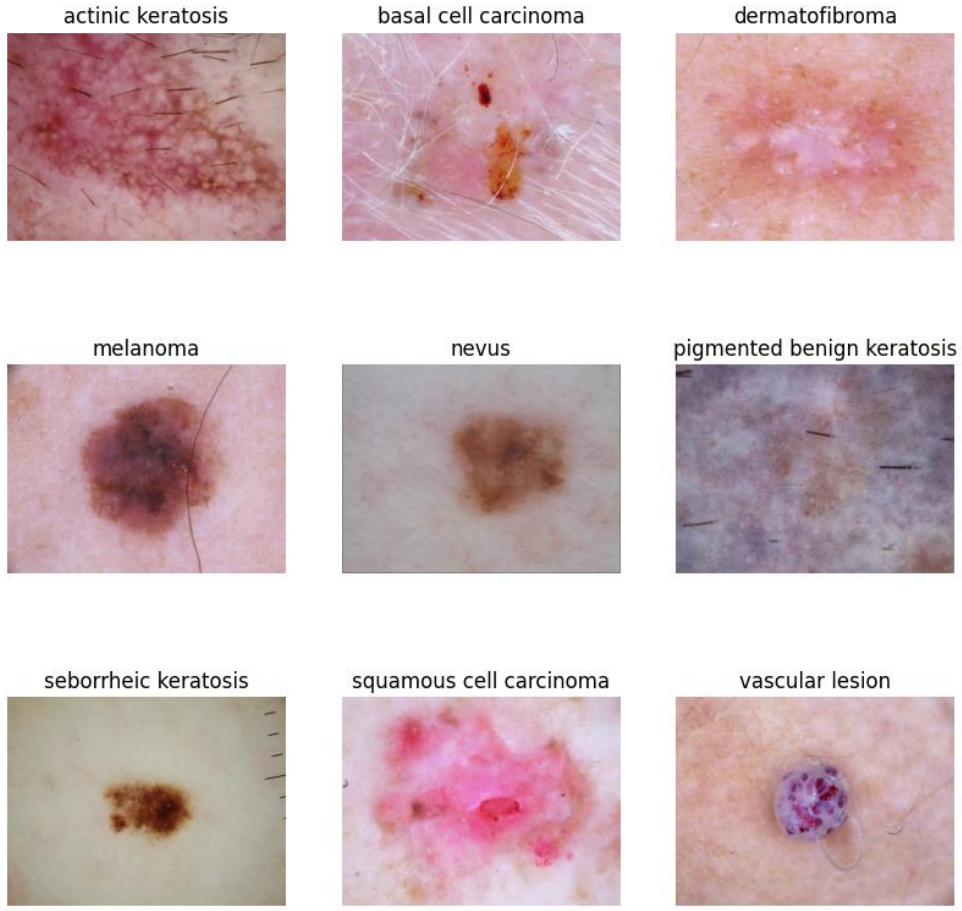
```
[47]  1 !fusermount -u drive

[48]  1 drive.mount("/content/drive/MyDrive/", force_remount = True)

      Mounted at /content/drive/MyDrive/

[65]  1 # Defining the path for train and test images
      2 data_dir_train = pathlib.Path("/content/drive/MyDrive/MyDrive/DATASETS/melanoma_skin_cancer_detection_dataset/Train")
      3 data_dir_test = pathlib.Path("/content/drive/MyDrive/MyDrive/DATASETS/melanoma_skin_cancer_detection_dataset/Test")
```



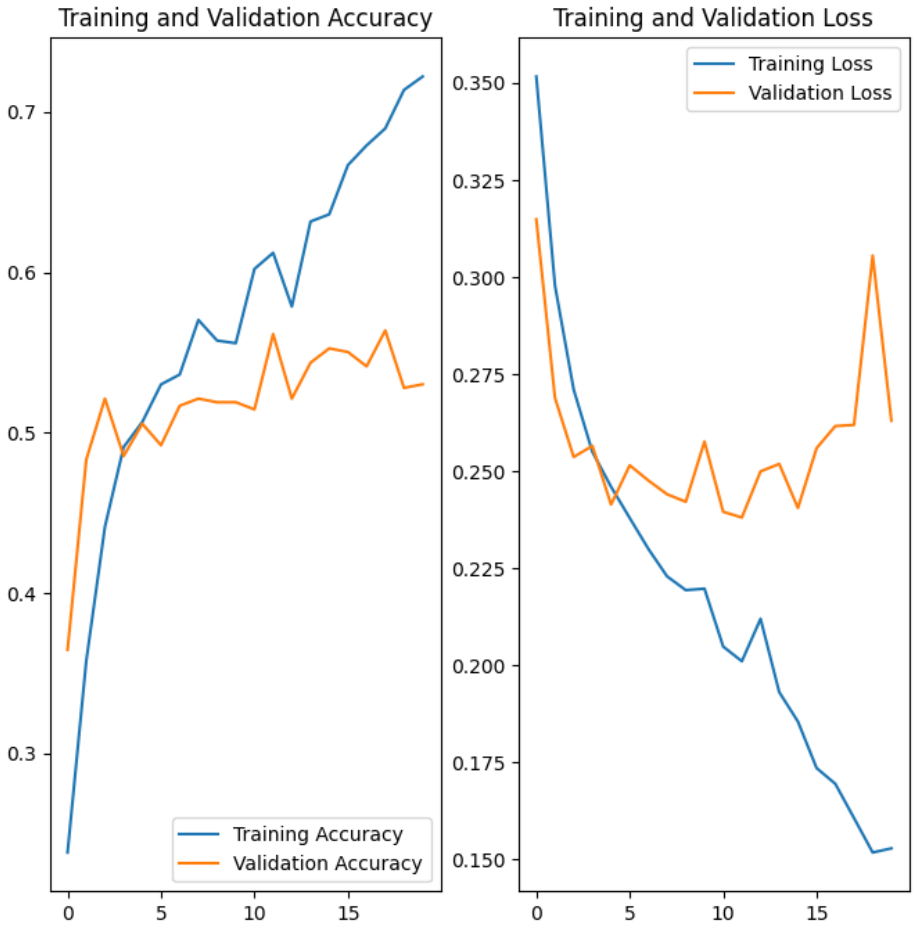
# Getting 1 image from each class and checking distribution

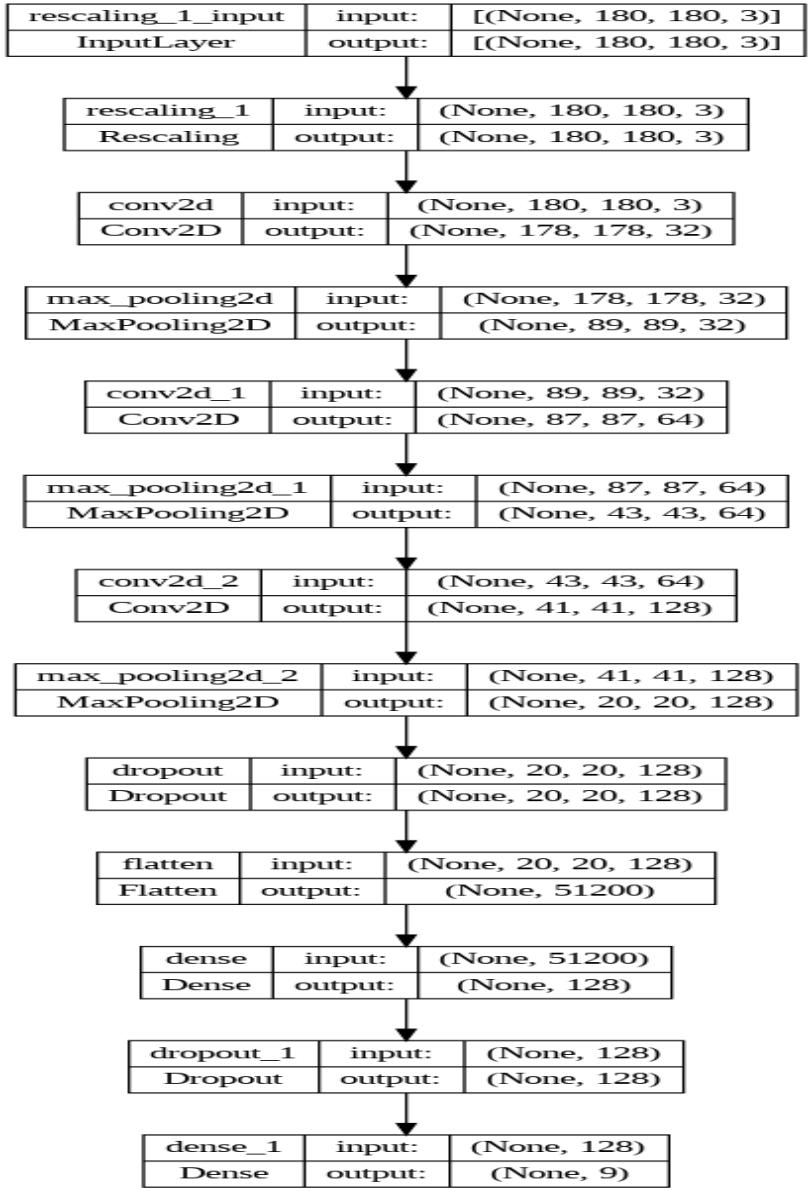
| | Class | No. of Image |
|---|---|---|
| 0 | basal cell carcinoma | 376 |
| 1 | nevus | 357 |
| 2 | melanoma | 438 |
| 3 | dermatofibroma | 95 |
| 4 | seborrheic keratosis | 77 |
| 5 | actinic keratosis | 114 |
| 6 | squamous cell carcinoma | 181 |
| 7 | vascular lesion | 139 |
| 8 | pigmented benign keratosis | 462 |



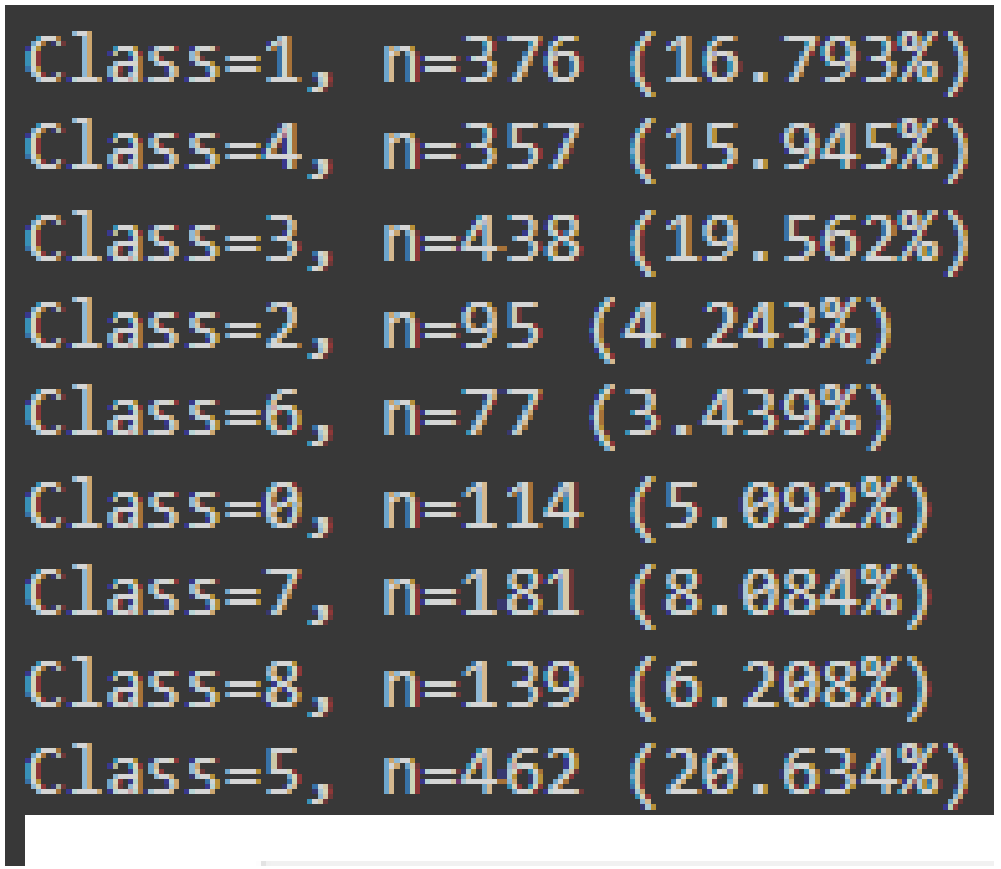
# Baseline optimized model (Next: Observations)



# Baseline Model Observations

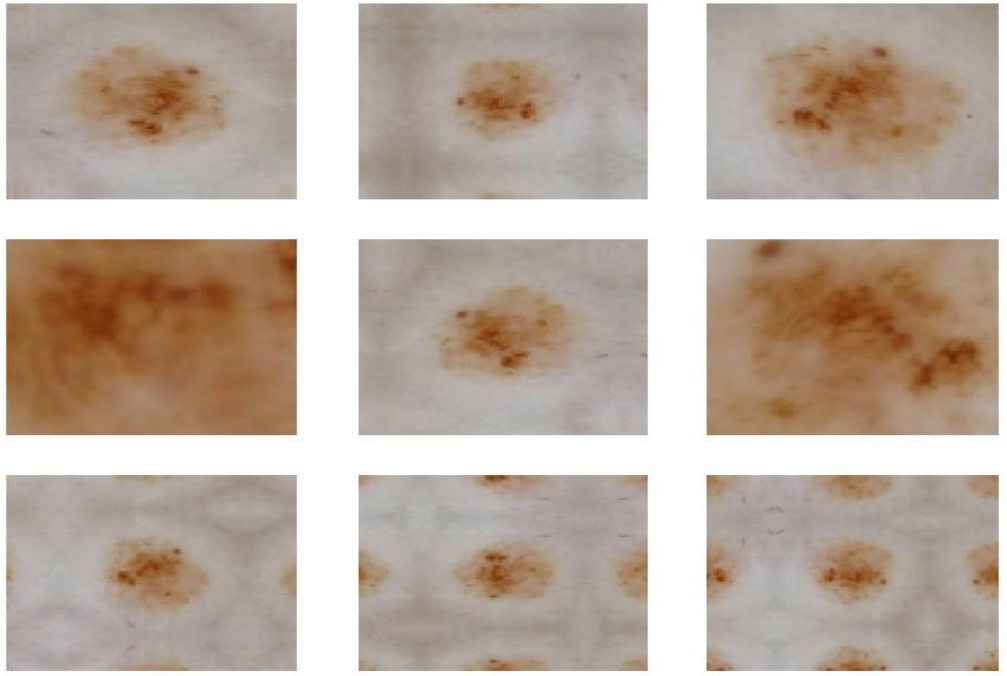
- From the distributions slide, it is clear actinic keratosis, seborrheic keratosis and dermatofibroma make up the minority classes while dominant classes include melanoma, pigmented benign keratosis, basal cell carcinoma and nevus majorly
- Image beside shows percentage distribution of the same
- The optimized baseline model overfits with over 70% training accuracy and over 50% validation accuracy
- Hence augmentation is required.



# Augmentation (without Augmentor)

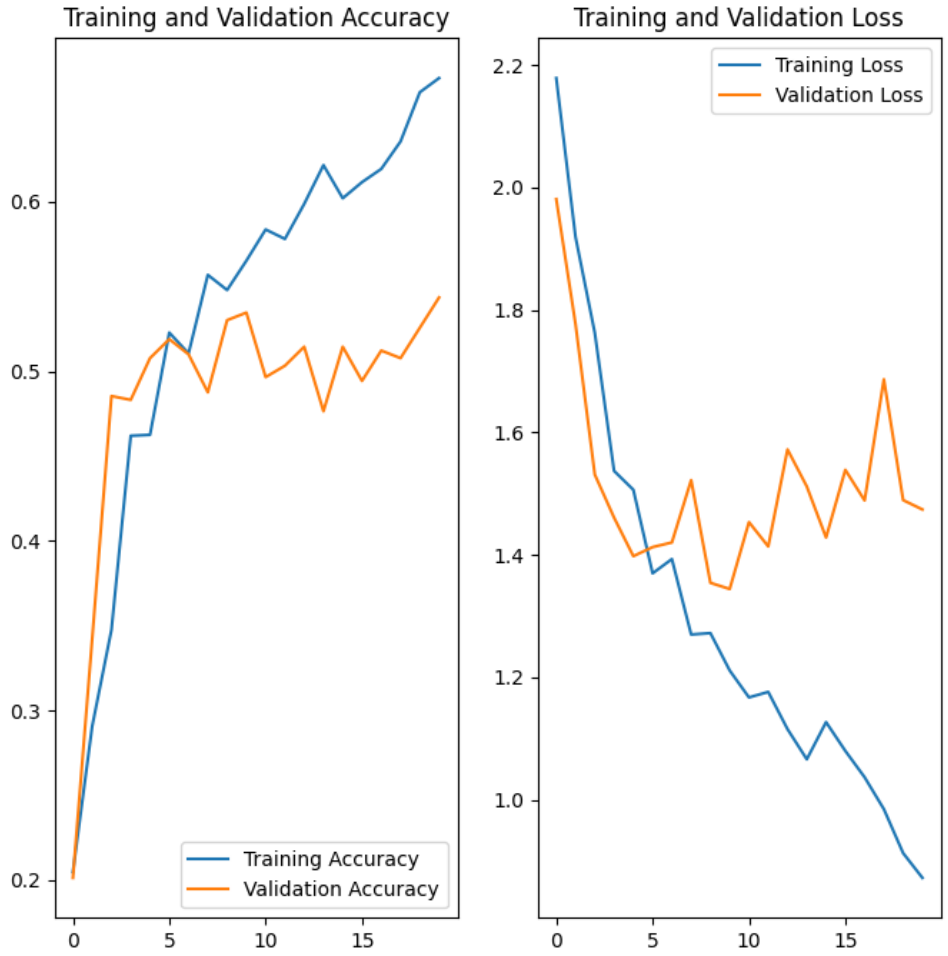
- This is basic augmentation and its effect on one of the image to understand what Augmentation does
- This was used on as dataset to train the same baseline model whose performance is discussed in next slide

```
data_augmentation = keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal",
                                                 input_shape=(img_height,
                                                              img_width,
                                                              3)),
    layers.experimental.preprocessing.RandomRotation(0.9),
    layers.experimental.preprocessing.RandomZoom(0.9),
  ]
)
```



# Performance after Basic Augmentation

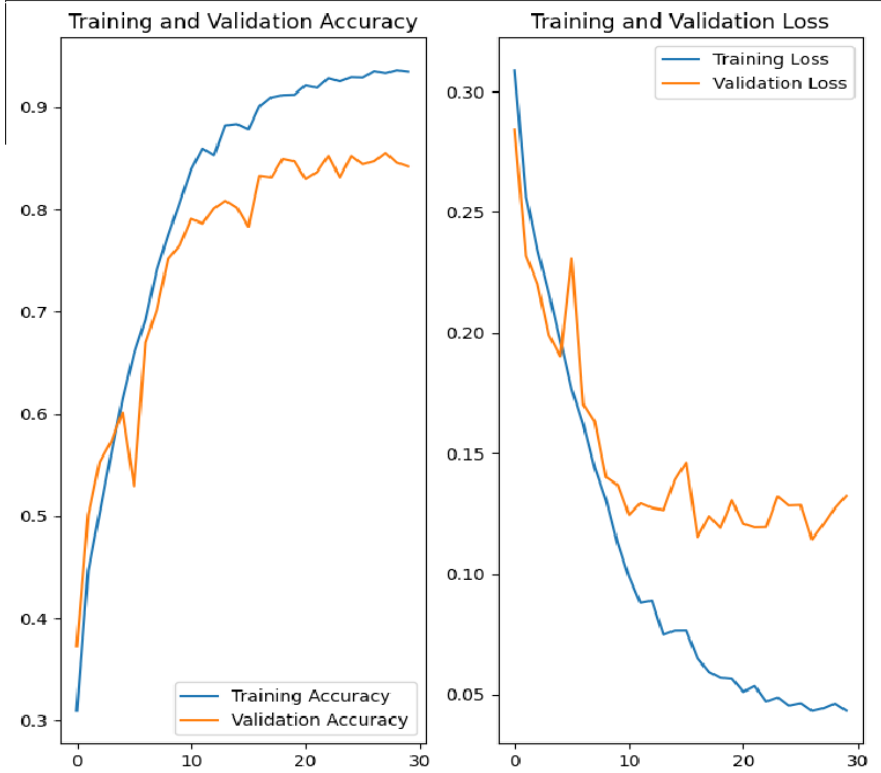
- As we see the overfitting reduced a bit after the primary augmentation
- The train model accuracy is still ~76% while validation accuracy came out to be ~54%
- So, we are on right path but need to change the augmentation methodology



# Augmentor Library use and Model Performance

- Using Augmentor, we added 500 samples to each class
- This reduced the class imbalance making minority classes less sparse
- The model performed better with training accuracy of ~96% and validation accuracy of ~85%
- Hence, overfit reduced to about half and performance increased
- So, our strategy was correct

```
!pip install Augmentor
path_to_training_dataset="/content/drive/MyDrive/MyDrive/Melonema_Case_Study/Datasets/Train/"
import Augmentor
for i in class_names:
    p = Augmentor.Pipeline(path_to_training_dataset + i)
    p.rotate(probability=0.7, max_left_rotation=10, max_right_rotation=10)
    p.sample(500)  #Adding 500 samples per class to make sure that none of the classes are sparse
```



# BONUS: Prediction on one image

- The code shows prediction on single image using the model
- As we tried on random sample images from the test dataset, we found, the model is predicting correctly on the samples chosen
- Herewith is one such image output along with prediction and actual labels

```
Test_image_path = os.path.join(data_dir_test, class_names[7], '*')
Test_image = glob(Test_image_path)
Test_image = load_img(Test_image[-1],target_size=(180,180,3))
plt.imshow(Test_image)
plt.grid(False)

img = np.expand_dims(Test_image,axis=0)
pred = model.predict(img)
pred = np.argmax(pred)
pred_class = class_names[pred]
print("Actual Class "+ class_names[7] +'\n'+ "Predicted Class "+pred_class )
```

```
1/1 [==============================] - 0s 255ms/step
Actual Class squamous cell carcinoma
Predicted Class squamous cell carcinoma
```