

VISION ASSIGNMENT REPORT

TOPIC:

**DEEP LEARNING BASED SKIN
CANCER DETECTION**

1. PITCH:

The early detection of Skin Cancer has been the serious concern for dermatologist clinics and hospitals. The Skin Cancer has become so common that according to internet resources more than 2 people die due to the skin cancer in the United States. Melanoma is a Skin Cancer which is 13th most common cancer in Men and it is 15th most common cancer in Women. In 2020, 150,000 cases were reported globally. And for early diagnosis, we need some technology that can assist doctors. That's why to provide a solution to this skin cancer diagnosis, I have proposed a project "Skin Cancer Detection using Deep Learning". This Deep Learning based model will assist dermatologists by providing second opinion on potential Skin Cancer cases. This Deep Learning based Skin Cancer Detection Model will reduce costs spent on misdiagnosis and unnecessary treatments. It will also provide faster diagnosis which will increase customer's trust and will also be helpful for timely diagnosis of such a deadly disease. This Deep Learning based Model will assist dermatologists in automating preliminary screenings and allowing them to focus on complex cases. This technology will be a breakthrough in Skin Cancer detection as well as it will pave way for other cancer diagnosis treatments to be done by Machine Learning based models in the future.

2. DATA SOURCE:

I have used data named as "Skin Cancer: Malignant vs Benign". I got this dataset from Kaggle. I found this dataset useful as it has two (2) classes to be precise:

- a) Malignant which is Cancerous.
- b) Benign which is Non-Cancerous.

The link to this dataset is: [Kaggle Dataset link](#)

3. MODEL AND DATA JUSTIFICATION:

My Justification for Model and Dataset which I have used in my Vision Assignment is below:

a. Model Justification:

I used **EfficientNet-B0** Model for Deep Learning based Skin Cancer Detection. The reason to chose this Model is the accuracy it provided me. I tried different Models like a simple CNN Model and AlexNet Model but the training accuracy I was getting after fine tuning was only around 80% for the Alexnet and Testing accuracy was even lower than this. And when I implemented my fine tuned EfficientNet-B0 model, I got 99.77% training accuracy and 90.45% testing accuracy. Which shows that EfficientNet-B0 was the best performer among simple CNN Model and AlexNet Model. That is why I preferred to use EfficientNet-B0 for my Vision Assignment.

b. Data Justification:

I opted to use this Dataset "Skin Cancer: Malignant vs Benign" because it has two folders as two classes that are Malignant for Cancerous class and Benign for Non-Cancerous class. This Dataset assisted my approach in providing diagnosis for Skin Cancer. The Model

based on this Dataset predicts whether the given image of Skin lies in Cancerous class or Non-Cancerous class.

Constraint: As the Model is trained on Skin Pictures for Cancer disease so it only takes image of skin and shows good performance on the skin Images. If any other image is provided to test the model for example image of chair, lock, flower, sky, or any other object except human skin then it is understood that you will get non-obvious and incorrect results.

4. COMMENTED EXAMPLES:

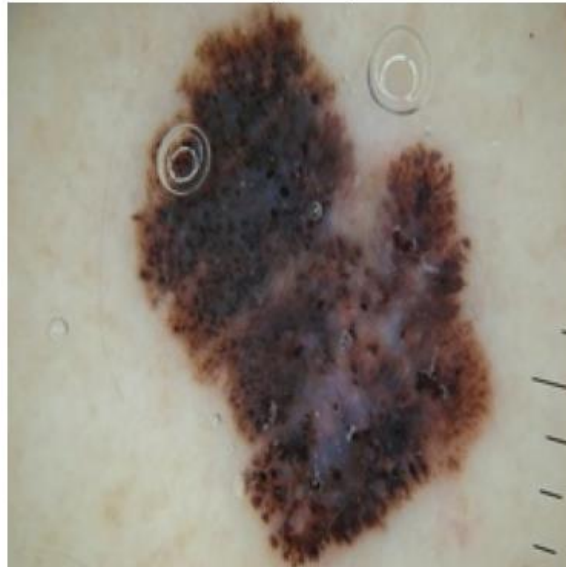
Training Example:

```
➡ Final training loss: 0.010  
Final validation loss: 0.506  
Final training accuracy: 99.77%  
Final validation accuracy: 85.30%
```

Here for Training, I got 99.77% accuracy and the loss was 1% which was I wanted. I wanted my Model to be trained perfectly on my Dataset and I got that result according to my satisfaction. So yes, I am satisfied with the training procedure.

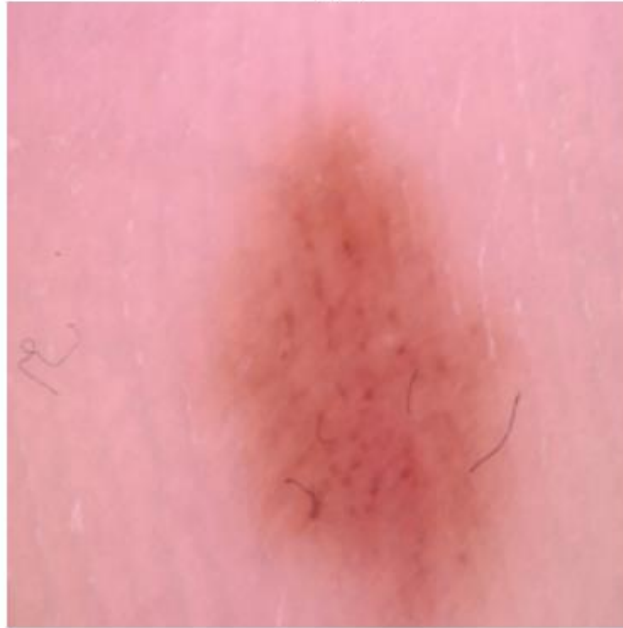
Testing Example:

Predicted Label: Malignant, Confidence: 1.00



This picture was actually from the Malignant class and my Model predicted it well. I set the parameter for confidence that if a given skin picture is predicted for Malignant class, then the confidence should be closer to **1**. Because in terms of probability, the value lies between **0 and 1**. And similarly for an image predicted to belong from Benign class should have the confidence closer to **0**. In below image you will see that I got confidence value exactly **0** which shows that my Model is performing very efficiently on the given inputs.

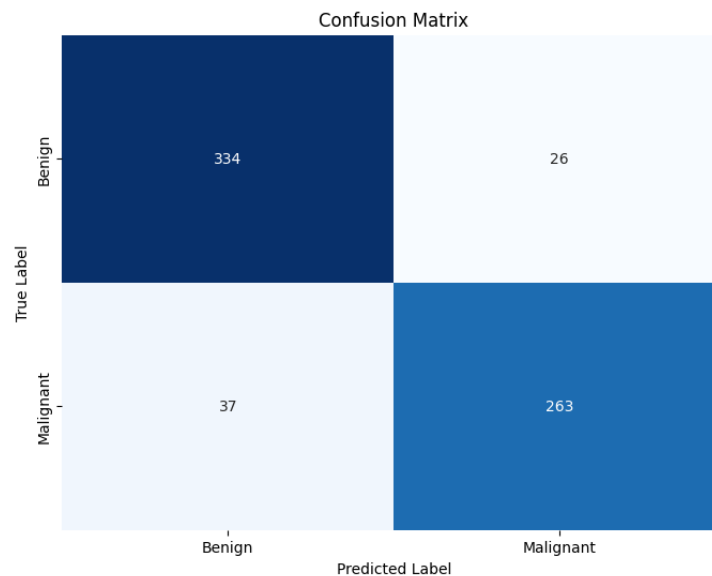
Predicted Label: Benign, Confidence: 0.00



The Image was actually from Benign class and it was predicted correctly as well. So yes, I am satisfied with the training as well.

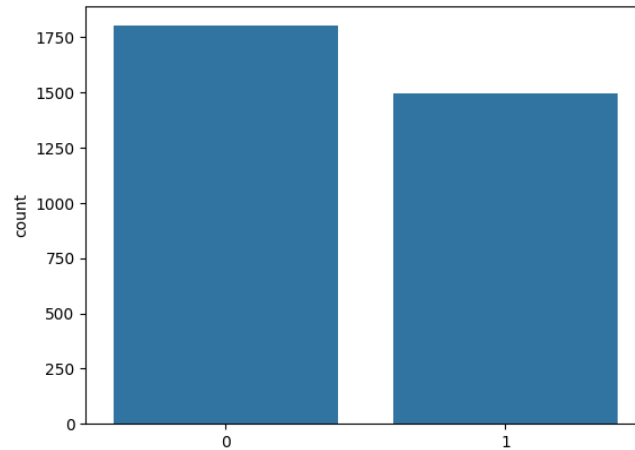
5. TESTING:

The Confusion matrix is Given below:



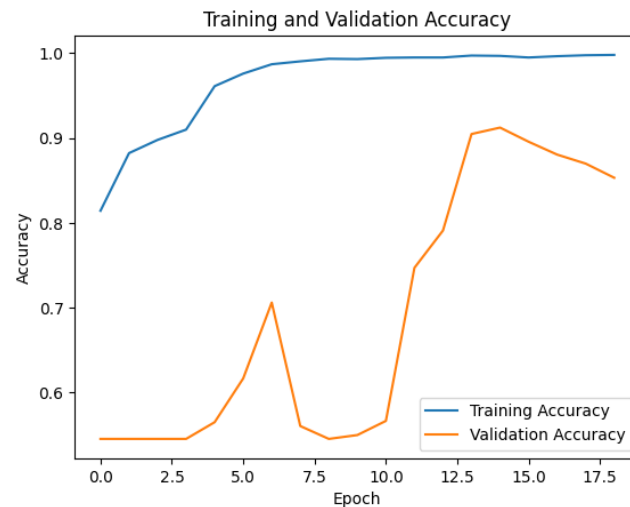
This Confusion Matrix shows the Model Prediction. The Model Predicted **597** Images correctly. Whereas on 63 occasions, it misclassified. The Misclassified predictions are just 9.54% of the total images included for testing. The Model predicted correctly for **90.46%** of the total images included for testing. Which showcase the efficient performance of the Model.

Data Frequency for Two Classes:



This histogram shows that the dataset has **1800** images from **Benign** class (including training and testing images) and **1497** images from **Malignant** class (including training and testing images).

Training and Validation Accuracy:



For my EfficientNet-B0 Model, the Training accuracy was **99.77%** whereas the Validation accuracy was **85.30%**. And the bar shows that the highest validation accuracy achieved was **91.21%** on 15th Epoch. I initially ran it for 50 Epochs but I placed the monitor for Epoch accuracy which stops the model once the maximum accuracy is achieved and if the further Epoch is reducing the accuracy, then it **early stops** the Epochs as seen below in our case.

```
Epoch 17/50
83/83 [=====] - 22s 269ms/step - loss: 0.0126 - accuracy: 0.9962 - val_loss: 0.4301 - val_accuracy: 0.8803 - lr: 1.0000e-05
Epoch 18/50
83/83 [=====] - ETA: 0s - loss: 0.0099 - accuracy: 0.9973
Epoch 18: ReduceLROnPlateau reducing learning rate to 1e-05.
83/83 [=====] - 22s 270ms/step - loss: 0.0099 - accuracy: 0.9973 - val_loss: 0.5343 - val_accuracy: 0.8697 - lr: 1.0000e-05
Epoch 19/50
83/83 [=====] - ETA: 0s - loss: 0.0097 - accuracy: 0.9977Restoring model weights from the end of the best epoch: 14.
83/83 [=====] - 22s 271ms/step - loss: 0.0097 - accuracy: 0.9977 - val_loss: 0.5059 - val_accuracy: 0.8530 - lr: 1.0000e-05
Epoch 19: early stopping
```

6. CODE AND INSTRUCTION TO RUN IT:

Code Link:

The code for Deep Learning Based Skin Cancer Detection can be accessed at the link given below:

[Skin Cancer Detection \(ipynb\)](#)

Model Diagram:

The model diagram for EfficientNet-B0 is given below:

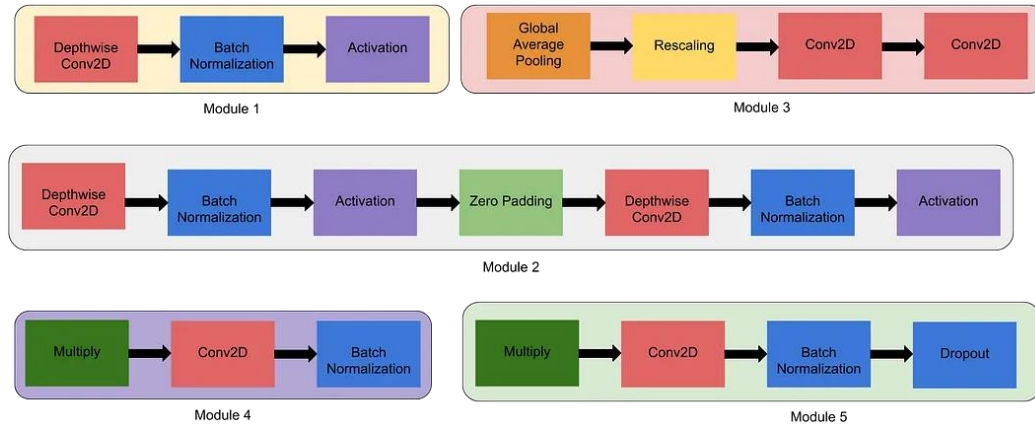


Figure: Modules in EfficientNet-B0

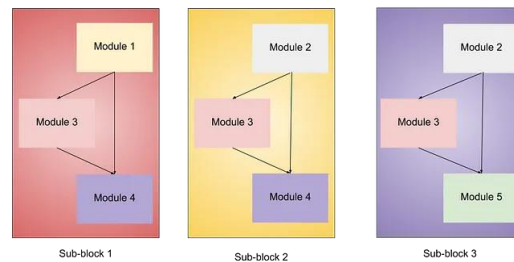


Figure: Sub-Modules in EfficientNet-B0

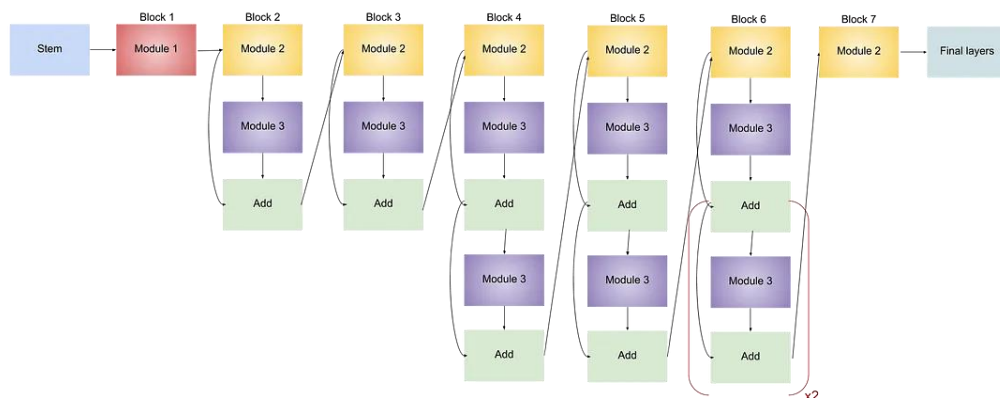


Figure: Model Diagram of EfficientNet-B0

Instructions to Run Code:

The Step-by-step procedure and guide to run the code is given below:

1. Download the Code file from the link. You will get the python Notebook when you download that file.
2. Open your Google Colab and upload the code file (python notebook) you just downloaded.
3. If you want to download and use data from the Kaggle through Kaggle API then run below code and Sign In to your Kaggle Account.

✓ DOWNLOAD DATASET FROM KAGGLE

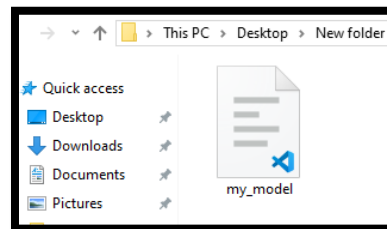
```
[ ] import opendatasets as od
url = 'https://www.kaggle.com/datasets/fanconic/skin-cancer-malignant-vs-benign/data'
od.download(url)
```

⚙ Skipping, found downloaded files in "./skin-cancer-malignant-vs-benign" (use force=True to force download)

Or you can download the dataset in your system by clicking the dataset link in this document at the start. After the dataset is downloaded to your system, provide correct path of train and test folder of the downloaded dataset in below shown code and then run this code.

```
[ ] path_train = '/content/skin-cancer-malignant-vs-benign/train'
path_test = '/content/skin-cancer-malignant-vs-benign/test'
```

4. You were provided with the file named as my_model.h5 file which a file having saved model in it. By using this file, you don't need to train your model again and you can just go on for testing the model based on this saved model.



Saved Model File

5. Upload this **my_model.h5** file to your Google Drive. Open Your Google Drive, click on upload and then upload this file.
6. When this file is uploaded, run the below code to connect to your google drive:

✓ CONNECTING GOOGLE DRIVE

```
[ ] from google.colab import drive
drive.mount('/content/drive')
```

⚙ Mounted at /content/drive

When you run this code, A screen will pop up to link you to your google drive. Select account and then It will ask you permission to what to do, Just click on Select All and then scroll to click Continue and your google drive will be connected to your colab environment.

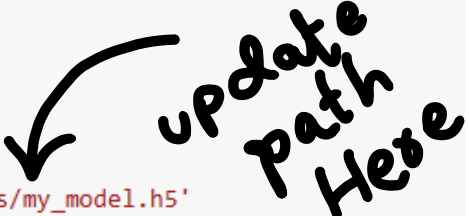
7. After Connecting to Google Drive, Click on left side bar where the folder sign is shown below key sign in Google Colab. And from here navigate to the file of saved model you uploaded named as **my_model.h5** file. Copy it's path from your drive and paste it in below code:

✓ MODEL TESTING

```
[ ] import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Load the trained model
model_path = '/content/drive/MyDrive/My Models/my_model.h5'
loaded_model = tf.keras.models.load_model(model_path)

# Load the image
image_path = '/content/drive/MyDrive/My Models/3.jpg'
image = tf.keras.preprocessing.image.load_img(image_path, target_size=(224, 224))
image_array = tf.keras.preprocessing.image.img_to_array(image)
image_array = np.expand_dims(image_array, axis=0) # Expand dimensions to create a batch
```



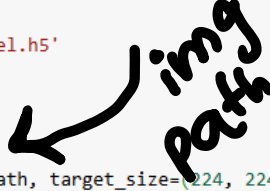
8. Similarly upload pics from the dataset downloaded from Kaggle to your Google Drive. You do not need to upload all pics, just pics you want to test for the prediction. And similarly navigate to the picture of skin you want to predict and copy it's path and paste it in below code where arrow is indicated:

✓ MODEL TESTING

```
[ ] import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Load the trained model
model_path = '/content/drive/MyDrive/My Models/my_model.h5'
loaded_model = tf.keras.models.load_model(model_path)

# Load the image
image_path = '/content/drive/MyDrive/My Models/3.jpg'
image = tf.keras.preprocessing.image.load_img(image_path, target_size=(224, 224))
image_array = tf.keras.preprocessing.image.img_to_array(image)
image_array = np.expand_dims(image_array, axis=0) # Expand dimensions to create a batch
```



9. Once all the paths are updated, Run the code by pressing **Shift + Enter**.
10. The Model will Predict the class of the given image.