Supervised By: Dr.Ahmed Youssef Badawy
Eng.Nour Eldeen Magdy
Machine Learning 2024

Project Name

Track :Medical Diagnosis

Skin Cancer Using Tensor Flow

Team Members:

Ali Emad el deen
Seif Tarek Mohamed
Youssef Ahmed Mosleh
Youssef Medhat Hanna

ABSTRACT

Skin cancer is among the most common types of cancer, and quick identification considerably enhances the odds of survival. The purpose of this work is to develop cutting-edge deep learning models that can classify images of skin cells and accurately **detect** cases of **skin cancer.**

Challenges and Problem Statements

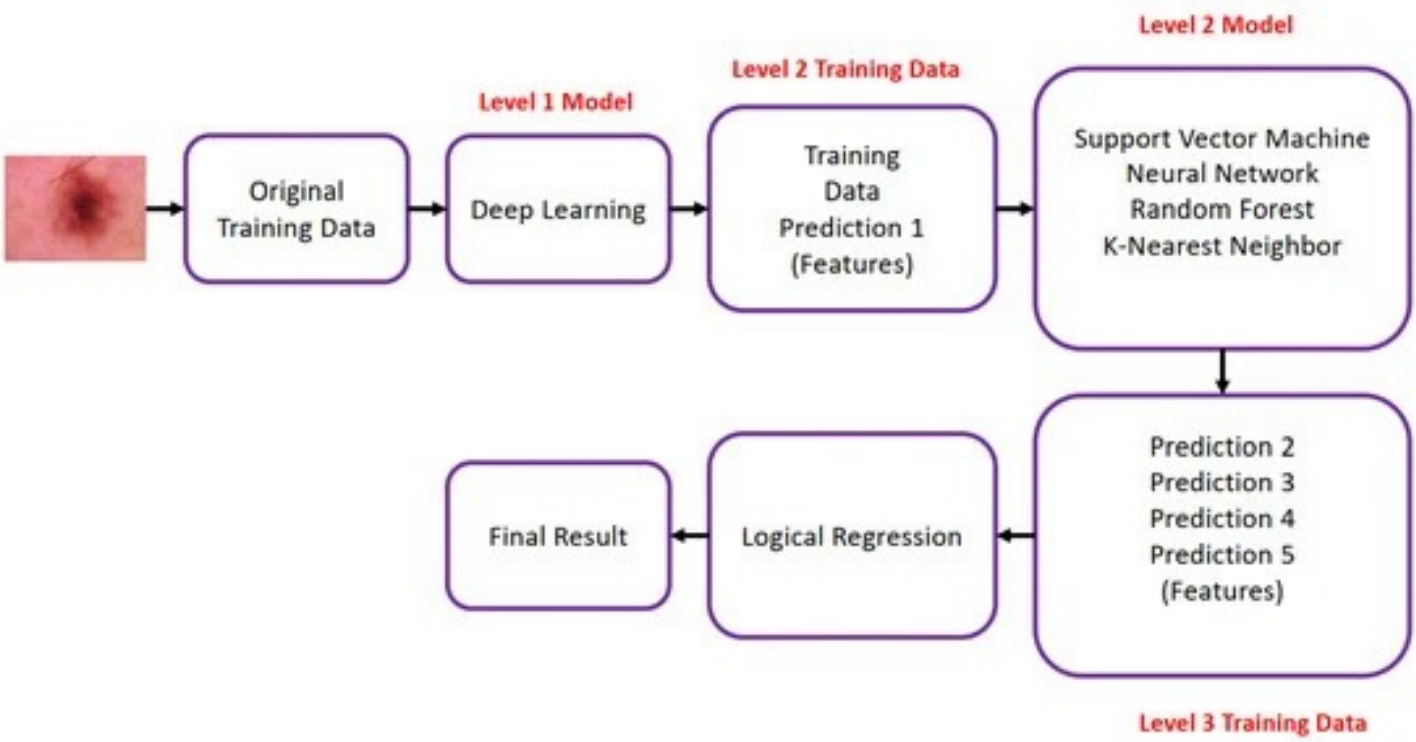
The principle barriers to skin cancer detection are that **it is a low priority in primary care, that the majority of exams do not result in significant findings, and that many providers lack expertise to adequately identify high risk lesions.So The main goal of using machine learning/deep learning in skin cancer classification is to improve the diagnostic accuracy of skin cancers,**

Project Overview

We will use a dataset that contains images for the two categories that are malignant or benign. We will use the transfer learning technique to achieve better results in less amount of training. We will use EfficientNet architecture as the backbone of our model along with the pre-trained weights of the same obtained by training it on the image net dataset.

Project Objectives

- Deep learning models can recognize complex pictures, text, sounds, and other data patterns to **produce accurate insights and predictions**.
- Specify whether Given Data set Images is benign or mallignant
- Algorithm:



Importing Libraries

Python libraries make it very easy for us to handle the data and perform typical and complex tasks with a single line of code.

- **Pandas**– This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.
- **Numpy**_– Numpy arrays are very fast and can perform large computations in a very short time.
- **Matplotlib**_– This library is used to draw visualizations.
- **Sklearn** – This module contains multiple libraries having pre-implemented functions to perform tasks from data preprocessing to model development and evaluation.
- **Tensor Flow**– This is an open-source library that is used for Machine Learning and Artificial intelligence and provides a range of functions to achieve complex functionalities with single lines of code.

```
[ ]  import numpy as np
     import pandas as pd
     import seaborn as sb
     import matplotlib.pyplot as plt

     from glob import glob
     from PIL import Image
     from sklearn.model_selection import train_test_split

     import tensorflow as tf
     from tensorflow import keras
     from keras import layers
     from functools import partial

     AUTO = tf.data.experimental.AUTOTUNE
     import warnings
     warnings.filterwarnings('ignore')
```

```
[ ]  images = glob('train_cancer/*/*.jpg')
     len(images)
```

Methodology

Now, let’s check the number of images we have got here,You can download Images from here train_cancer - Google Drive

```
#replace backslash with forward slash to avoid unexpected errors
images = [path.replace('\\', '/') for path in images]
df = pd.DataFrame({'C:\Users\youss\OneDrive\Desktop\ML Project\DS.zip\train_cancer': images})
df['label'] = df['C:\Users\youss\OneDrive\Desktop\ML Project\DS.zip\train_cancer'].str.split('/', expand=True)[1]
df.head()
```

	filepath	label
0	train_cancer/benign/30.jpg	benign
1	train_cancer/benign/19.jpg	benign
2	train_cancer/benign/25.jpg	benign
3	train_cancer/benign/14.jpg	benign
4	train_cancer/benign/41.jpg	benign

```
[ ] images = glob('train_cancer/*/*.jpg')
    len(images)
```

Converting the labels to 0 and 1 will save our work of label encoding

```
df['label_bin'] = np.where(df['label'].values == 'malignant', 1, 0)
df.head()
```

```
x = df['label'].value_counts()
plt.pie(x.values,
    labels=x.index,
    autopct='%1.1f%%')
plt.show()
```

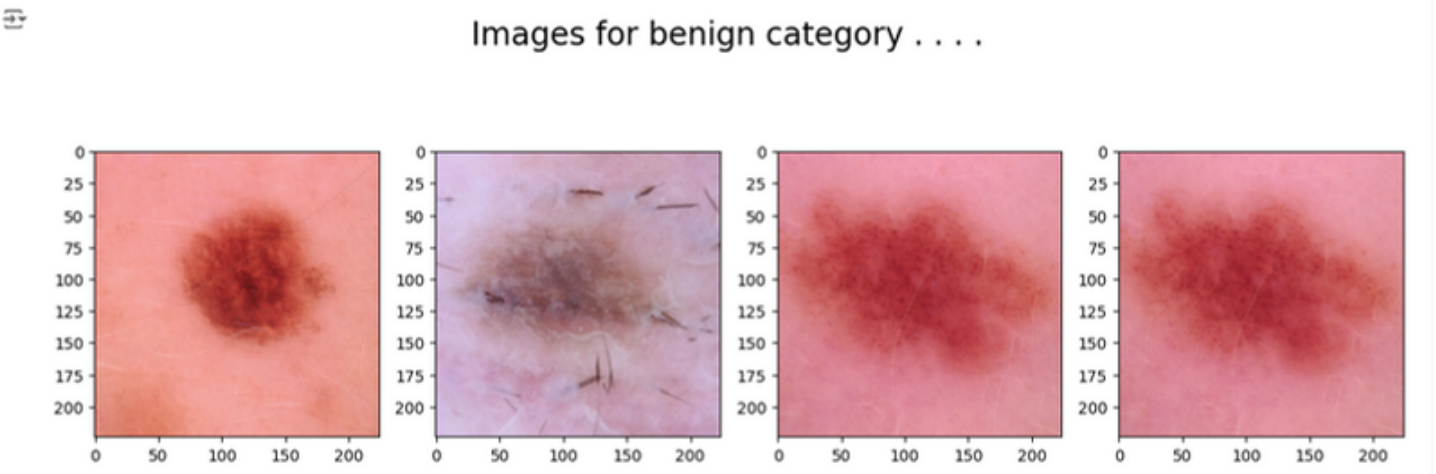
An approximately equal number of images have been given for each of the classes so, data imbalance is not a problem here.

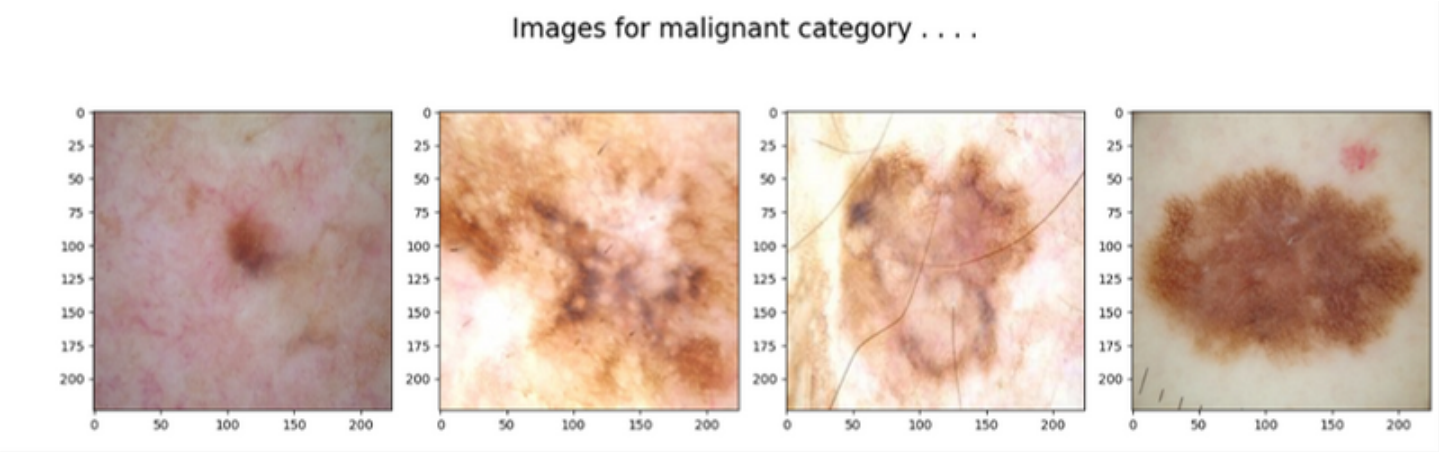
```
for cat in df['label'].unique():
  temp = df[df['label'] == cat]

  index_list = temp.index
  fig, ax = plt.subplots(1, 4, figsize=(15, 5))
  fig.suptitle(f'Images for {cat} category . . . .', fontsize=20)
  for i in range(4):
    index = np.random.randint(0, len(index_list))
    index = index_list[index]
    data = df.iloc[index]

    image_path = data[0]

    img = np.array(Image.open(image_path))
    ax[i].imshow(img)
plt.tight_layout()
plt.show()
```





Now, let's split the data into training and validation parts by using the train_test_split function.

```
features = df['C:\Users\youss\OneDrive\Desktop\ML Project\DS.zip\train_cancer']
target = df['label_bin']

X_train, X_val, Y_train, Y_val = train_test_split(features, target,
                  test_size=0.15,
                  random_state=10)

X_train.shape, X_val.shape
```

```
((229,), (41,))
```

```
def decode_image(C:\Users\youss\OneDrive\Desktop\ML Project\DS.zip\train_cancer, label=None):

  img = tf.io.read_file(filepath)
  img = tf.image.decode_jpeg(img)
  img = tf.image.resize(img, [224, 224])
  img = tf.cast(img, tf.float32) / 255.0

  if label == 0:
    Label = 0
  else:
    Label = 1
```

Image input pipelines have been implemented below so, that we can pass them without any need to load all the data beforehand.

```
train_ds = (
  tf.data.Dataset
  .from_tensor_slices((X_train, Y_train))
  .map(decode_image, num_parallel_calls=AUTO)
  .batch(32)
  .prefetch(AUTO)
)

val_ds = (
  tf.data.Dataset
  .from_tensor_slices((X_val, Y_val))
  .map(decode_image, num_parallel_calls=AUTO)
  .batch(32)
  .prefetch(AUTO)
)
```

```
[ ] from tensorflow.keras.applications.efficientnet import EfficientNetB7

    pre_trained_model = EfficientNetB7(
      input_shape=(224, 224, 3),
      weights='imagenet',
      include_top=False
    )

    for layer in pre_trained_model.layers:
      layer.trainable = False
```

```
from tensorflow.keras import Model

inputs = layers.Input(shape=(224, 224, 3))
x = layers.Flatten()(inputs)

x = layers.Dense(256, activation='relu')(x)
x = layers.BatchNormalization()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.3)(x)
x = layers.BatchNormalization()(x)
outputs = layers.Dense(1, activation='sigmoid')(x)

model = Model(inputs, outputs)
```

```
[ ] model.compile(
      loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
      optimizer='adam',
      metrics=['AUC']
    )
```

```
[ ] history = model.fit(train_ds,
              validation_data=val_ds,
              epochs=5,
              verbose=1)
```

```
history = model.fit(train_ds,
          validation_data=val_ds,
          epochs=5,
          verbose=1)
```

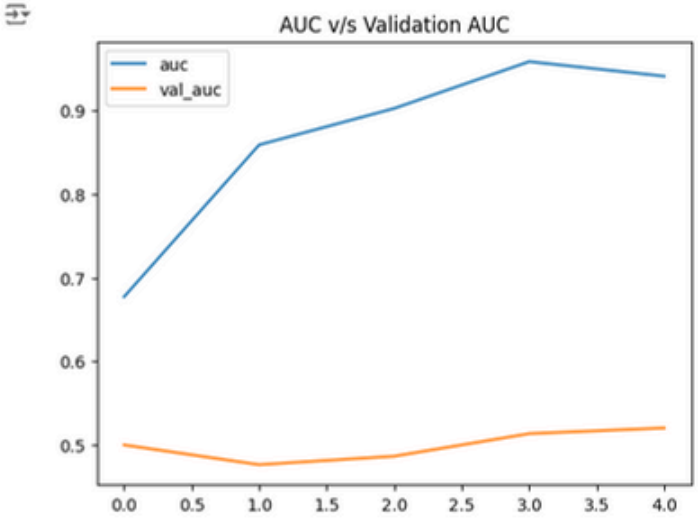
```
Epoch 1/5
8/8 [==============================] - 8s 745ms/step - loss: 0.8845 - auc: 0.6772 - val_loss: 2.4704 - val_auc: 0.5000
Epoch 2/5
8/8 [==============================] - 6s 697ms/step - loss: 0.6240 - auc: 0.8589 - val_loss: 0.8174 - val_auc: 0.4764
Epoch 3/5
8/8 [==============================] - 6s 693ms/step - loss: 0.5195 - auc: 0.9025 - val_loss: 1.2005 - val_auc: 0.4865
Epoch 4/5
8/8 [==============================] - 6s 723ms/step - loss: 0.4439 - auc: 0.9585 - val_loss: 0.7520 - val_auc: 0.5135
Epoch 5/5
8/8 [==============================] - 7s 925ms/step - loss: 0.3778 - auc: 0.9411 - val_loss: 0.6217 - val_auc: 0.5203
```

```
[ ] hist_df = pd.DataFrame(history.history)
    hist_df.head()
```

	loss	auc	val_loss	val_auc
0	0.884477	0.677150	2.470377	0.500000
1	0.624001	0.858943	0.817437	0.476351
2	0.519542	0.902520	1.200533	0.486486
3	0.443939	0.958507	0.752024	0.513514
4	0.377754	0.941076	0.621685	0.520270

```
hist_df['loss'].plot()
hist_df['val_loss'].plot()
plt.title('Loss v/s Validation Loss')
plt.legend()
plt.show()
```

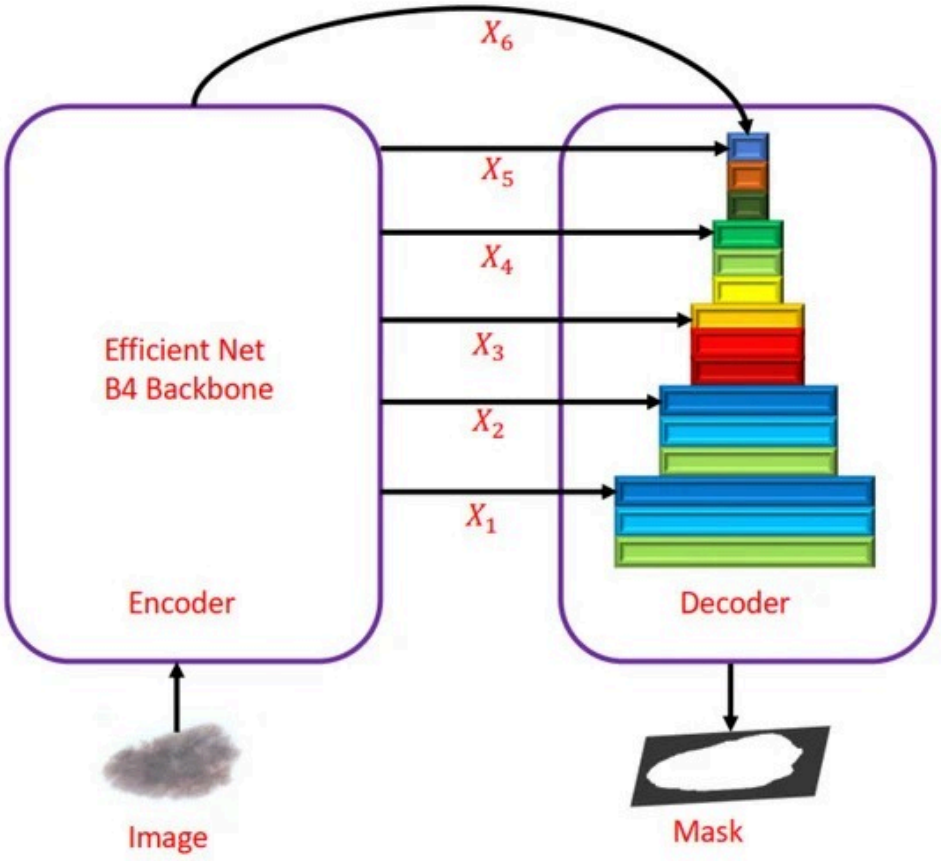

```
hist_df['auc'].plot()
hist_df['val_auc'].plot()
plt.title('AUC v/s Validation AUC')
plt.legend()
plt.show()
```



Model Development:

For this task, we will use the EfficientNet architecture and leverage the benefit of pre-trained weights of such large networks.

Model Architecture:



We will implement a model using the Functional API of Keras which will contain the following parts:

- The base model is the EfficientNet model in this case.
- The Flatten layer flattens the output of the base model's output.
- Then we will have two fully connected layers followed by the output of the flattened layer.
- We have included some Batch Normaliztion layers to enable stable and fast training and a Dropout layer before the final layer to avoid any possibility of overfitting.
- The final layer is the output layer which outputs soft probabilities for the three classes.

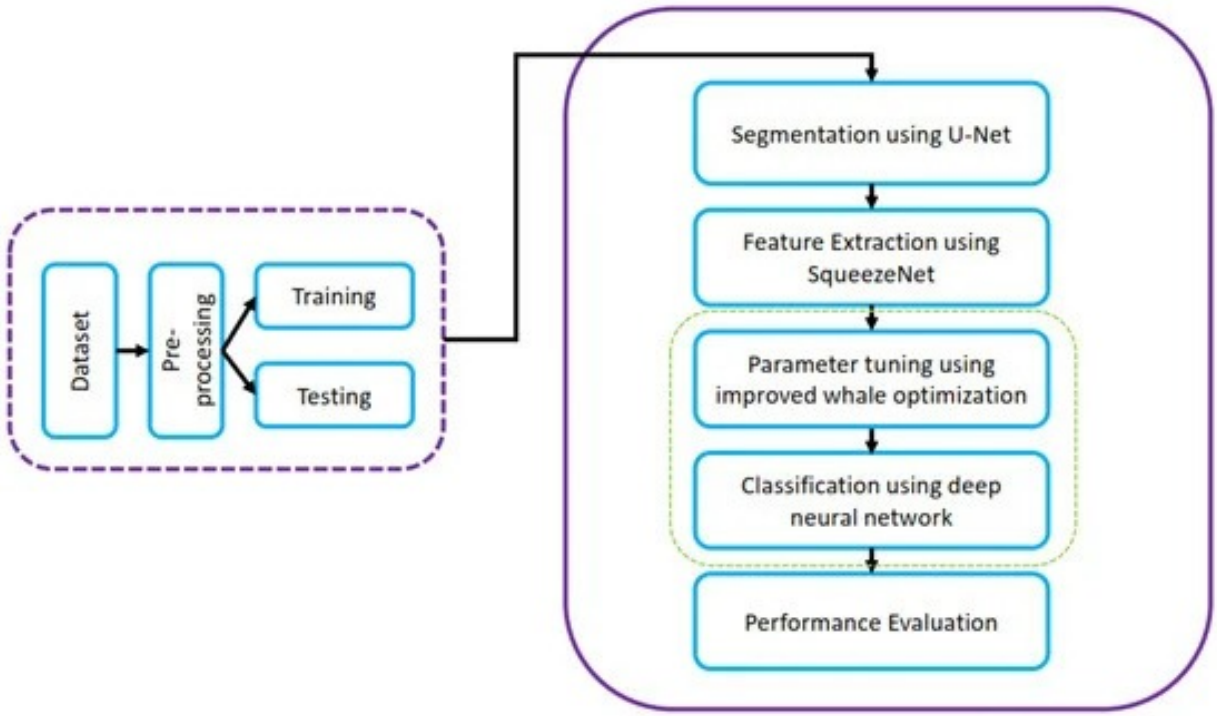
Output:

While compiling a model we provide these three essential parameters:

- **optimizer** – This is the method that helps to optimize the cost function by using gradient descent.
- **loss** – The loss function by which we monitor whether the model is improving with training or not.

- **metrics** – This helps to evaluate the model by predicting the training and the validation data.

BLOCK DIAGRAM



Industry Relevance

To facilitate the development of computer-aided diagnosis systems in the segmentation and classification of melanoma. To build and evaluate the MED-NODE system for detecting skin cancer with dermoscopic images. To address the small size and insufficient diversity of images in the skin-disease dataset.

Busineess & Entrepreneurship Potential

TensorFlow Optimizations from Intel:

Intel collaborates with Google* to upstream most optimizations into the stock distribution of TensorFlow with the newest optimizations and features being released earlier as **Intel® Extension for TensorFlow***. These optimizations can be enabled with a few lines of code and will accelerate TensorFlow-based training and **inference** performance on Intel CPU and GPU hardware.

Intel Neural Compressor:

Intel Neural Compressor is an open-source Python library that runs on CPUs or GPUs that performs model quantization to reduce the model size and increase the speed of deep learning inference for deployment. Intel Neural Compressor is a part of the Intel AI Analytics Toolkit. This library automates popular methods such as quantization, compression, pruning, and knowledge distillation across multiple deep learning frameworks.

References

1.Research Gate.com

2.MDPI

3.geeks for greeks.com

- For Notebook: Google Colab
- For Dataset: train cancer


THANK YOU