SKINMENT

AI based End-to-end Android application which detect specific mole is benign or malignant

Skinment

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF ENGINEERING
IN
COMPUTER ENGINEERING

Submitted By:
Project Group 06
Jan 2021- May 2021

THIRD YEAR, COMPUTER ENGINEERING
ANJUMAN-I-ISLAM KALSEKAR TECHNICAL CAMPUS

SUBMITTED TO:
Prof. Aamer Syed

SUBMITTED BY:
Sayed Mohd Kazim Mehdi / 18CO47
Sayyed Faraz Ali Akbar Ali /17CO42
Afif Rafique Pallavkar /18CO17
Syed Abuzar Ali Munawwar Ali /18CO16

Anjuman-I-Islam's Kalsekar Technical Campus
School of Engineering,

Plot No. 2 & 3, Sector - 16, Near Thana Naka,
Khandagao, New Panvel, Navi Mumbai, Maharashtra 410206

(AY 2020-21)

TEAM MEMBERS

- 18CO47 | Sayed Mohd Kazim Mehdi
- 17CO42 | Sayyed Faraz Ali Akbar Ali
- 18CO17 | Afif Rafique Pallavkar
- 18CO16 | Syed Abuzar Ali Munawwar Ali

DECLARATION

We hereby declare that the report work entitled "**Skinment** " submitted to the AIKTC, is a record of an original work done by us under the guidance of Prof. Aamer Syed, Assistant prof. Computer Engineering Department, Anjuman-I-Islam's Kalsekar Technical Campus. This project work is submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Engineering.
We further declare that the results embodied in this Report have not been submitted to any other University or Institute for the award of any degree or diploma.

AIKTC, New-Panvel.

SAYED MOHD KAZIM MEHDI
Team Lead: Skinment

Date : 10/05/2021

CERTIFICATE

This is to certify that the Report entitled "**Skinment**" is a record of bona fide work carried out by the Sayed Mohd Kazim Mehdi(18CO47) ,Sayyed Faraz Ali Akbar Ali(17CO42),Afif RafiquePallavkar(18CO17),Syed Abuzar Ali Munawwar Ali(18CO16) during the academic year 2020-2021 in partial fulfillment of the award of the degree of Bachelor of Engineering in Computer Engineering by the Kalsekar technical Campus, New-Panvel. They worked on this report under the supervision of Prof. Aamer Syed. During their tenure with us we found them sincere and hard working. We wish them great success in the future.

Date: 10/ 05/ 2021

Prof. Aamer Syed	*Prof. Tabrez Khan*	*Dr. Abdul Razak*
Supervizor	*HOD,Computer Engineering*	*Honnutagi*
		Director,AIKTC

Anjuman-I-Islam's
Kalsekar technical Campus, New Panvel

Mini Project Approval

This Mini Project entitled ”Skinment” by Name of Sayed Mohd Kazim Mehdi(18CO47) , Sayyed Faraz Ali Akbar Ali(17CO42), Afif Rafique Pallavkar(18CO17),Syed Abuzar Ali Munawwar Ali(18CO16) is approved for the degree of Bachelor of Engineering in Computer Engineering.

Examiners
1.....

(Internal Examiner Name & Sign)

2.....

(External Examiner name & Sign)

Date: 25/05/2021

Place: New Panvel, India

Table of Content

1.Introduction

What is Skin Cancer ?

Uncontrolled growth of abnormal skin cells. Often caused by ultraviolet radiation from sunshine or tanning beds.Potential Genetic basis for susceptibility. A major public health problem, with over 5 million newly diagnosed cases in the United States each year.

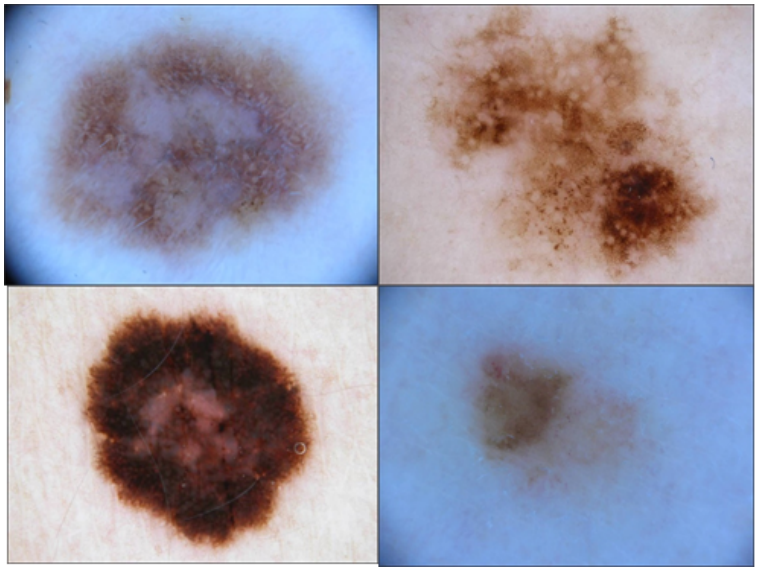
How do physicians diagnose skin cancer?

A skin examination by a dermatologist is the way to get a definitive diagnosis of skin cancer.In many cases, the appearance alone is sufficient to make the diagnosis.Melanoma is the deadliest form of skin cancer, responsible for over 9,000 deaths each year.

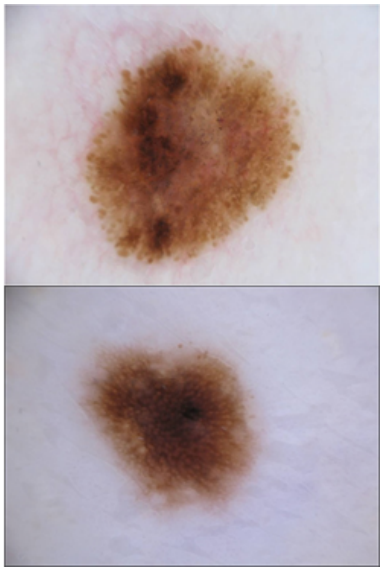
Solution that we proposed:

We propose a smartphone based skin cancer detection system that utilizes deep learning and low-cost camera to take the snapshots of suspected skin lesions and distinguish between malignant and benign melanoma skin images.

Malignant Samples



Benign Samples



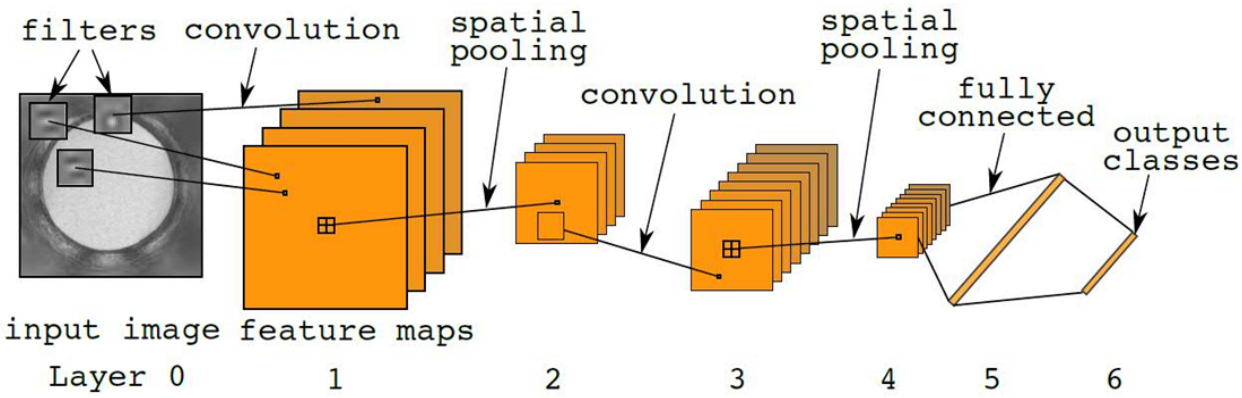
1.2 Data Source

Dataset collected from kaggle
A dataset of 2637 skin images was selected containing:
1440 images belonging to benign.
1198 images belonging to malignant.
Dataset was collected from here.

Acknowledgements

All the rights of the Data are bound to the ISIC-Archive rights (https://www.isic-archive.com/#!/topWithHeader/wideContentTop/main). I do not take any responsibility for the right-infringement of any kernels. Thus, do not monetize any of your models done on this data.

2. Architecture



There are two main parts to a CNN architecture
A convolution tool that separates and identifies the various features of the image for analysis in a process called Feature Extraction.

A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.

Convolutional Layer

This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).
The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

Pooling Layer

In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon the method used, there are several types of Pooling operations. In Max Pooling, the largest element is taken from the feature map. Average Pooling calculates the average of the elements in a predefined size Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer

Fully Connected Layer

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.

Dropout

Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model's performance when used on new data.

To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropout of 0.3, 30% of the nodes are dropped out randomly from the neural network.

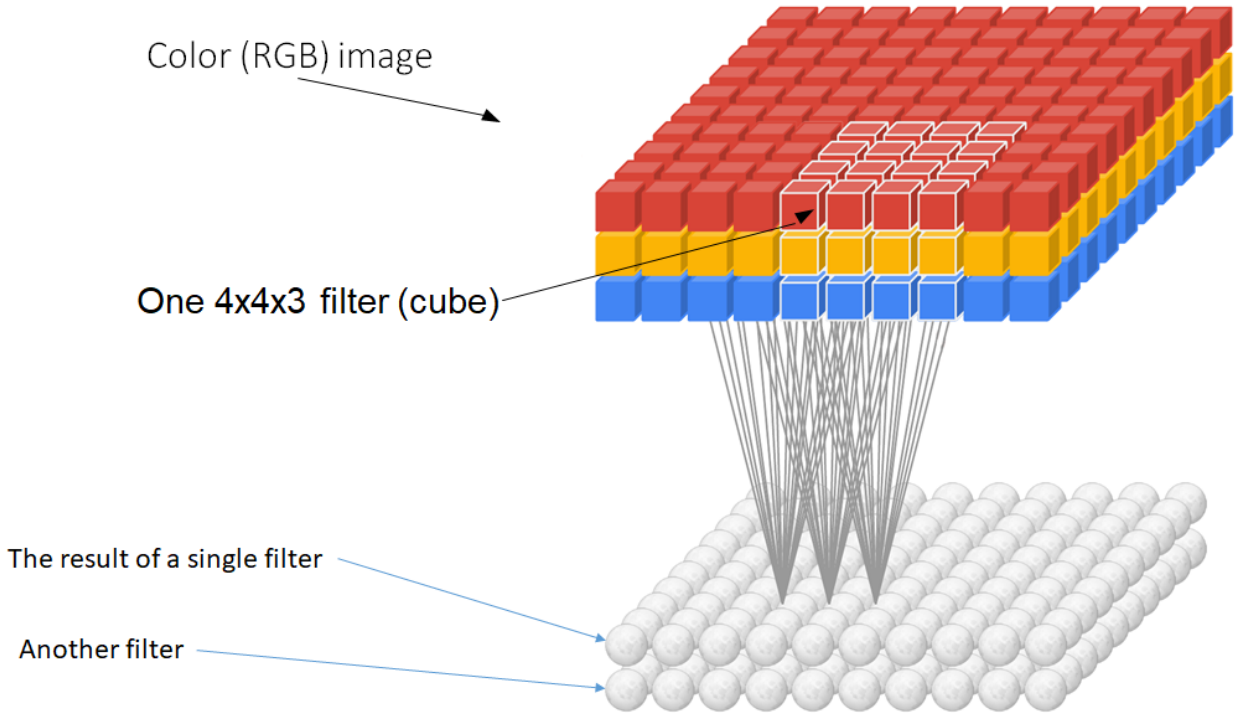
Activation Functions

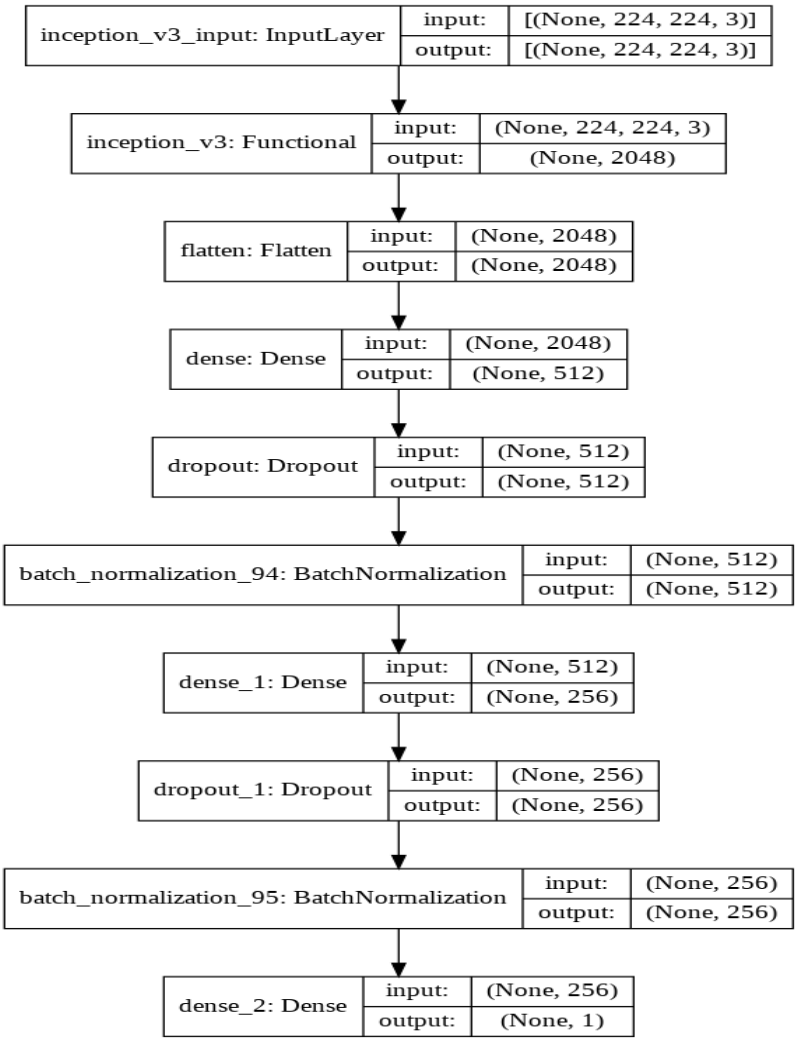
Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides

which information of the model should fire in the forward direction and which ones should not at the end of the network. It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred and for a multi-class classification, generally softmax is used.

2.1 Model Architecture

Input






```
model=Sequential()

pretrained_model=InceptionV3(include_top=False,
                             input_shape=(224, 224, 3),
                             pooling='avg',
                             weights='imagenet')
```

```
for layer in pretrained_model.layers:
if(isinstance(layer,tf.keras.layers.BatchNormalization)
):
        layer.trainable=True

model.add(pretrained_model)
model.add(Flatten())

model.add(Dense(512,activation='relu'))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Dense(256,activation='relu'))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Dense(1,activation='sigmoid'))
```

Inception v3

We adopt Inception Net, a deep Learning model trained on ImageNet dataset to classify natural images as our base model Training a model such as Inception Net can take even weeks on a high end GPU

Therefore we fine-tuned Inception Net for the task of skin lesion classification into Malignant and Benign Melanoma

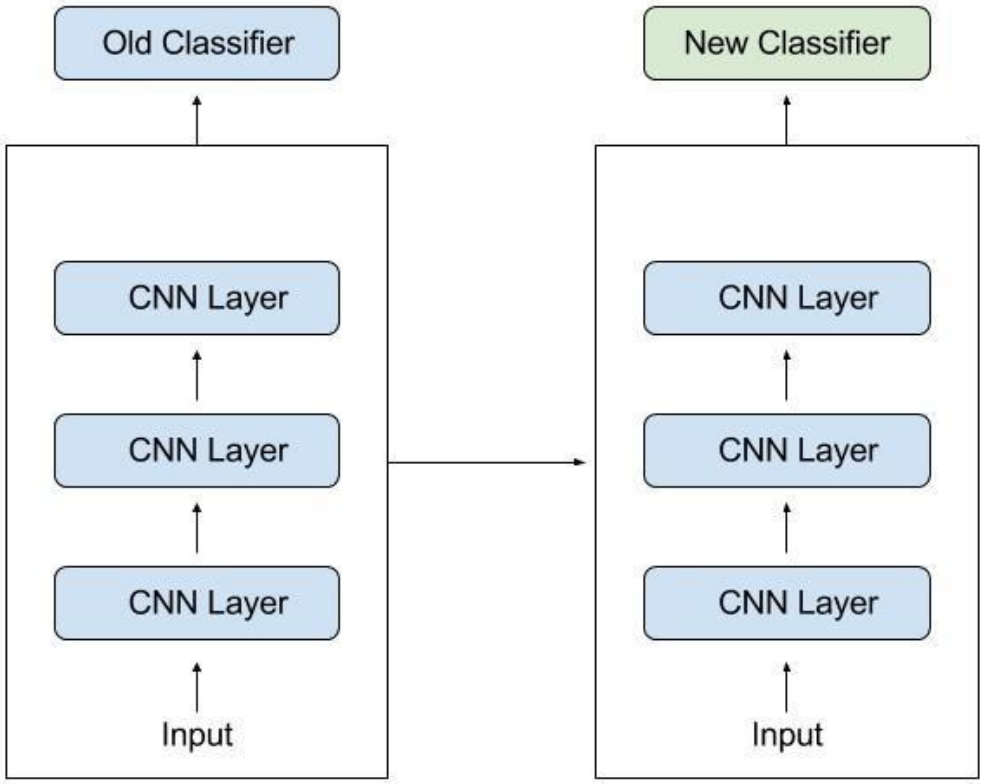
2.2 Transfer Learning

Transfer learning, used in machine learning, is the reuse of a pre-trained model on a new problem. In transfer learning, a machine exploits the knowledge gained from a previous task to improve generalization about another. For example, in training a classifier to predict whether an image contains food, you could use the knowledge it gained during training to recognize drinks.

HOW IT WORKS

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 labeled data of the task it was initially trained on.

Let’s go back to the example of a model trained for recognizing a backpack on an image, which will be used to identify sunglasses. In the earlier layers, the model has learned to recognize objects, because of that we will only retrain the latter layers so it will learn what separates sunglasses from other objects.



classifiers transfer learning
In transfer learning, we try to transfer as much knowledge as possible from the previous task the model was trained on to the new task at hand. This knowledge can be in various forms depending on the problem and the data. For example, it could be how models are composed, which allows us to more easily identify novel objects.

2.3 Why CNN?

Over the years, research on convolutional neural networks (CNNs) has progressed rapidly, however the real-world

deployment of these models is often limited by computing resources and memory constraints. What has also led to extensive research in ConvNets is the accuracy of difficult classification tasks that require understanding abstract concepts in images.

Another reason why CNN are hugely popular is because of their architecture — the best thing is there is no need for feature extraction. The system learns to do feature extraction and the core concept of CNN is, it uses convolution of image and filters to generate invariant features which are passed onto the next layer. The features in the next layer are convoluted with different filters to generate more invariant and abstract features and the process continues till one gets the final feature / output (let say face of X) which is invariant to occlusions.

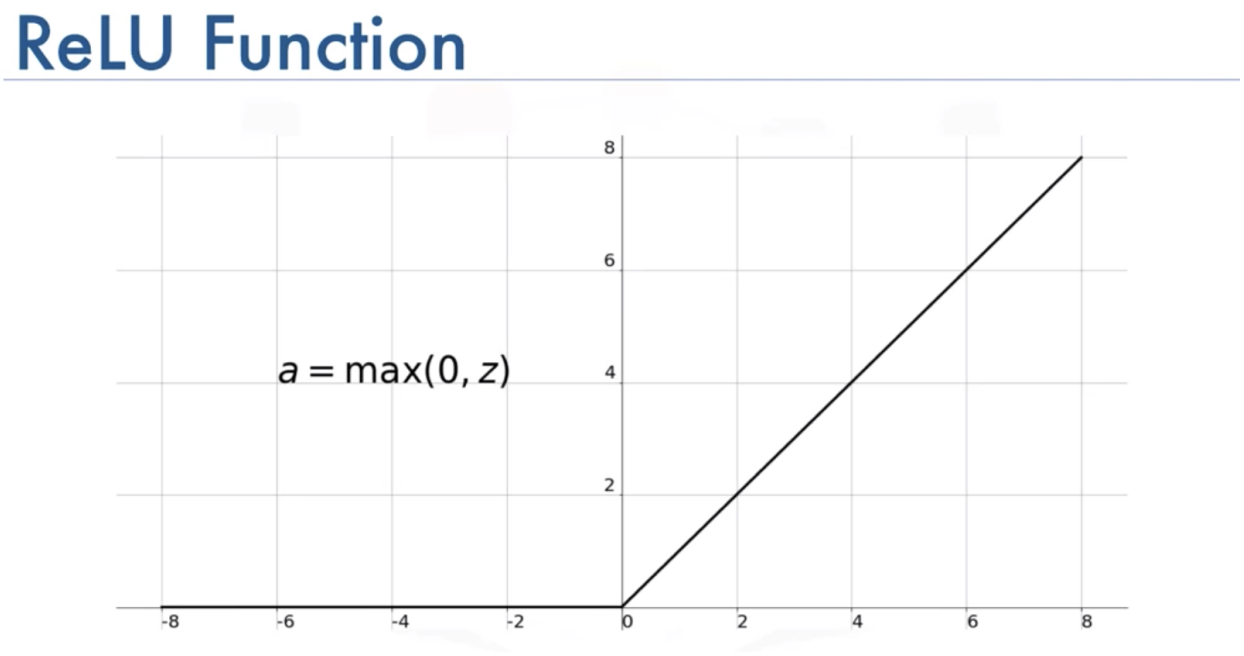
Also, another key feature is that deep convolutional networks are flexible and work well on image data. As one researcher points out, convolutional layers exploit the fact that an interesting pattern can occur in any region of the image, and regions are contiguous blocks of pixels. But one of the reasons why researchers are excited about deep learning is the potential for the model to learn useful features from raw data. Now, convolutional neural networks can extract informative features from images, eliminating the need of traditional manual image processing methods.

2.4 Relu and Softmax

Rectified Linear Units (ReLU)

ReLU : A Rectified Linear Unit (A unit employing the rectifier is also called a rectified linear unit ReLU) has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input. The operation of ReLU is closer to the way our biological neurons work.

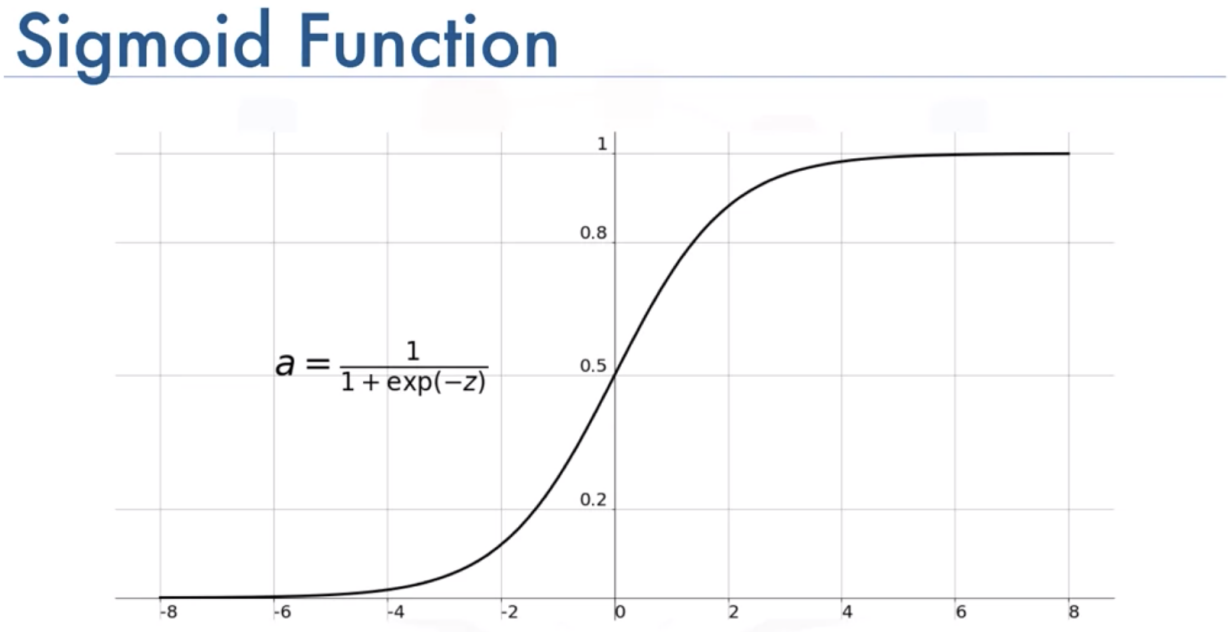
$$f(x) = max(x, 0)$$



Sigmoid Function

Sigmoid functions are used in machine learning for the logistic regression and basic neural network implementations and they are the introductory activation units.

$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$



3. Model for Edge Computing Devices

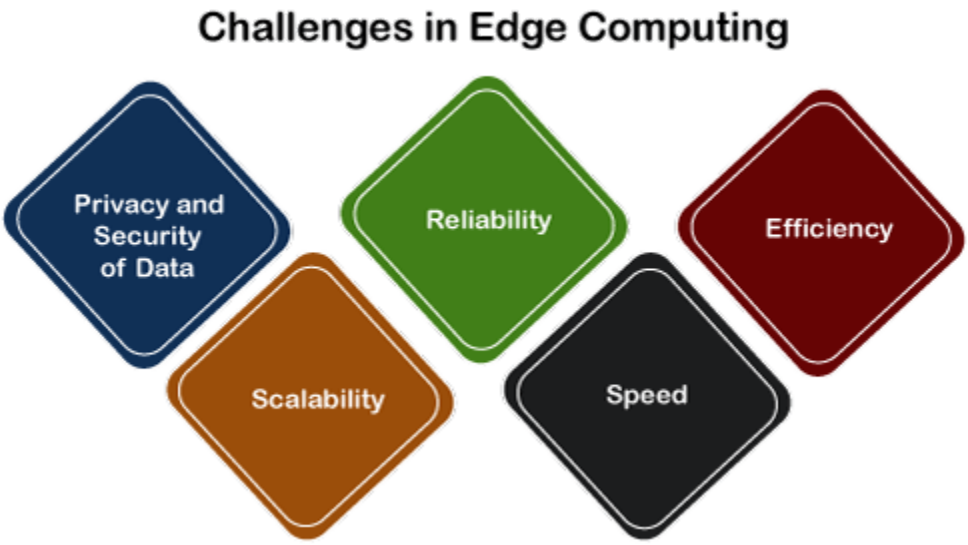
What is Edge Computing

Edge Computing is a buzzword such as cloud, IoT, and Artificial Intelligence. Simply saying, Edge Computing brings the decentralization of networks. Edge Computing is the upcoming enhancement and advancement in technology. The literal meaning of the word 'Edge' is the geographic location on the planet to deliver services in a distributed manner. Edge Computing is a distributed computing system that allows computation of data and storage too close to the source (where data is required). It brings computing as much closer as possible so as to minimize the bandwidth, improve response time, and use of latency. Instead of locating the data at a

centralized place, the concept of edge computing believes in distributing the computing process of the data.

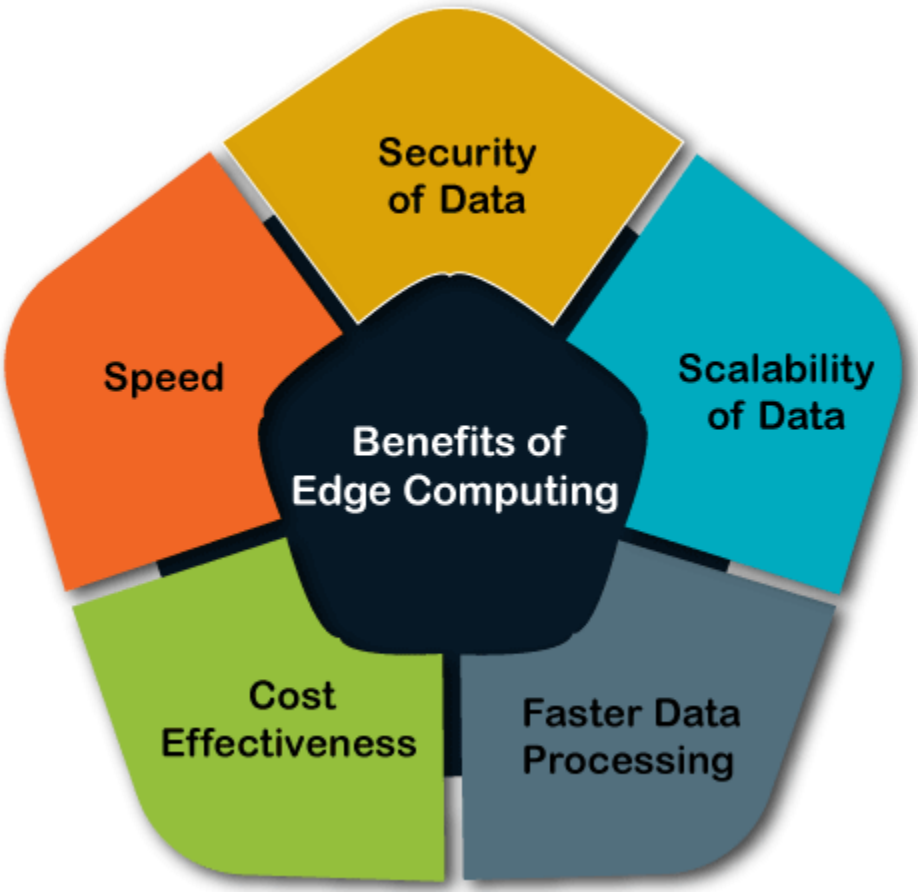
E.g. Mobile devices, Smartphones, IOT based devices etc

Challenges in Edge Computing



Applications of Edge Computing

Benefits of Edge Computing



3.1 Converting .h5 to tflite

End to End: tf.Keras to TFLite to Android

Train your model (tf.keras)
3 choices for model building:

- Sequential
- Functional
- Model subclassing

Convert to tflite (tf.lite.TFLiteconverter)
2 choices for conversion:

- Python code (recommended)
- Command line

Run on Android:

1. Model file under /assets
2. Update build.gradle
3. Input image
4. Preprocessing
5. Classify with the model
6. Post processing
7. Display result in UI

Python code:

```
import tensorflow as tf
from tensorflow import keras
import tensorflow_hub as hub

model_path='/content/model.h5'
model=keras.models.load_model(model_path)
reloaded =
keras.models.load_model(model_path,custom_objects{'Kera
sLayer':hub.KerasLayer})

TFLITE_MODEL = f"path/model.tflite"


# Get the concrete function from the Keras model.
```

```
run_model = tf.function(lambda x : reloaded(x))

# Save the concrete function.
concrete_func = run_model.get_concrete_function(
    tf.TensorSpec(model.inputs[0].shape,
model.inputs[0].dtype)
)

# Convert the model to standard TensorFlow Lite model
converter =
tf.lite.TFLiteConverter.from_concrete_functions([concre
te_func])
converted_tflite_model = converter.convert()
open(TFLITE_MODEL, "wb").write(converted_tflite_model)
```

3.2 Tflite interpreter in android

Platform: Android

The Java API for running an inference with TensorFlow Lite is primarily designed for use with Android, so it's available as an Android library dependency: `org.tensorflow:tensorflow-lite`.

In Java, you'll use the `Interpreter` class to load a model and drive model inference. In many cases, this may be the only API you need.

You can initialize an `Interpreter` using a `.tflite` file:

```
public Interpreter(@NotNull File modelFile);
```

Or with a `MappedByteBuffer`:

```
public Interpreter(@NotNull MappedByteBuffer
mappedByteBuffer);
```

In both cases, you must provide a valid TensorFlow Lite model or the API throws `IllegalArgumentException`. If you use `MappedByteBuffer` to initialize an `Interpreter`, it must remain unchanged for the whole lifetime of the `Interpreter`.

The preferred way to run inference on a model is to use signatures - Available for models converted starting Tensorflow 2.5

```
try (Interpreter interpreter = new
Interpreter(file_of_tensorflowlite_model)) {
  Map<String, Object> inputs = new HashMap<>();
  inputs.put("input_1", input1);
  inputs.put("input_2", input2);
  Map<String, Object> outputs = new HashMap<>();
  outputs.put("output_1", output1);
  interpreter.runSignature(inputs, outputs, "mySignature");
}
```

The `runSignature` method takes three arguments:

- **Inputs** : map for inputs from input name in the signature to an input object.
- **Outputs** : map for output mapping from output name in signature to output data.
- **Signature Name** [optional]: Signature name (Can be left empty if the model has a single signature).

Another way to run an inference when the model doesn't have a defined signature. Simply call `Interpreter.run()`. For example:

```
try (Interpreter interpreter = new
Interpreter(file_of_a_tensorflowlite_model)) {
  interpreter.run(input, output);
}
```

The `run()` method takes only one input and returns only one output. So if your model has multiple inputs or multiple outputs, instead use:

```
interpreter.runForMultipleInputsOutputs(inputs,
map_of_indices_to_outputs);
```

In this case, each entry in `inputs` corresponds to an input tensor and `map_of_indices_to_outputs` maps indices of output tensors to the corresponding output data.

In both cases, the tensor indices should correspond to the values you gave to the TensorFlow Lite Converter when you created the model. Be aware that the order of tensors in `input` must match the order given to the TensorFlow Lite Converter.

The `Interpreter` class also provides convenient functions for you to get the index of any model input or output using an operation name:

```
public int getInputIndex(String opName);
public int getOutputIndex(String opName);
```

If `opName` is not a valid operation in the model, it throws an `IllegalArgumentException`.

Also beware that `Interpreter` owns resources. To avoid memory leak, the resources must be released after use by:

```
interpreter.close();
```

4. Model in Android Application

Deployment

Deployment is the process of packaging and updating your ML model for use on Android when doing on-device inference. There are three options available:

Include the model with your Android app

Your model is deployed with your app like any other asset. Updates to the model require updating the app. There are two ways you can add a model to your app:

- Include the model in the app's APK file. This is the most basic option.
- Include the model in an Android App Bundle and use feature modules.

Provide the model at runtime

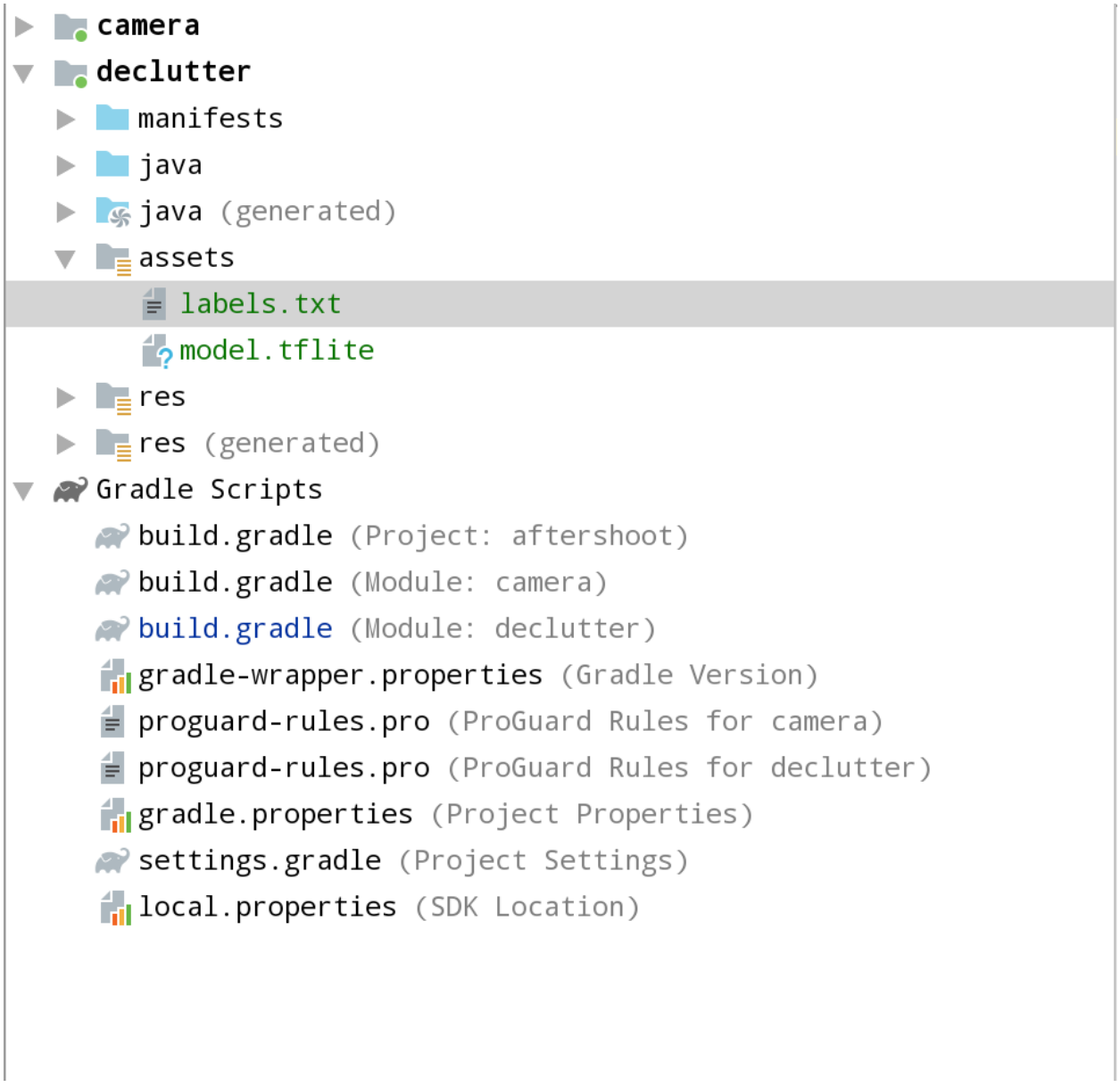
This enables you to update your model independently of your app. This also makes A/B testing easier. You can serve your custom model using the Firebase ML Custom Models function or host the model download with your own infrastructure.

A combination of both

It is not unusual for developers to package an initial version of their model with their Android app so that users does not need to wait for their model to download while updating the model to a new version.

4.1 How to use tflite model

To add the model, open Android Studio and select "*File -> New Folder -> Assets Folder*". This should create an assets folder in your app—move the tflite model to this folder, along with the labels.txt file containing your labels. Once done, it should look like this:



After this, open your app's build.gradle file and add the following dependency to your dependencies block:

```
dependencies {
  ...
  implementation 'org.tensorflow:tensorflow-lite:+'
  ...
}
```

Load the model file from assets

To actually use the model, you first need to load it from the assets folder. To do this, you can use the following method:

```
class MainActivity : AppCompatActivity(){
    ...
    private fun loadModelFile(): MappedByteBuffer? {
        val fileDescriptor: AssetFileDescriptor =
assets.openFd("model.tflite")
        val inputStream =
FileInputStream(fileDescriptor.fileDescriptor)
        val fileChannel: FileChannel = inputStream.channel
        val startOffset: Long = fileDescriptor.startOffset
        val declaredLength: Long =
fileDescriptor.declaredLength
        return
fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset,
declaredLength)
    }
    ...
}
```

The method above will read our tflite model and return a MappedByteBuffer object. We can then use this object to create our interpreter like so:

```
class MainActivity : AppCompatActivity(){
...
    val interpreter by lazy {
        Interpreter(loadModelFile())
    }

    private fun loadModelFile(): MappedByteBuffer {
        val fileDescriptor: AssetFileDescriptor =
assets.openFd("model.tflite")
        val inputStream =
FileInputStream(fileDescriptor.fileDescriptor)
        val fileChannel: FileChannel = inputStream.channel
        val startOffset: Long = fileDescriptor.startOffset
        val declaredLength: Long =
fileDescriptor.declaredLength
        return
fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset,
declaredLength)
    }
...
}
```

Preparing the input

Before we pass any image to this interpreter, it's important that we prepare the input accordingly. Since the model here accepts a single RGB image with dimensions 224 x 224 pixels, we need to resize our input bitmaps before we perform any inference on them.Before we go ahead, let's first initialize a few variables.

```
class MainActivity : AppCompatActivity(){
    ...
    // Our model expects a RGB image, hence the channel
size is 3
```

```
    private val channelSize = 3

    // Width of the image that our model expects
    var inputImageWidth = 224

    // Height of the image that our model expects
    var inputImageHeight = 224

    // Size of the input buffer size (if your model expects
a float input, multiply this with 4)
    private var modelInputSize = inputImageWidth *
inputImageHeight * channelSize

    // Output you get from your model, this is essentially
as we saw in netron
    val resultArray = Array(1) { ByteArray(3) }
    ...
}
```

Pass the input to the interpreter and get the output

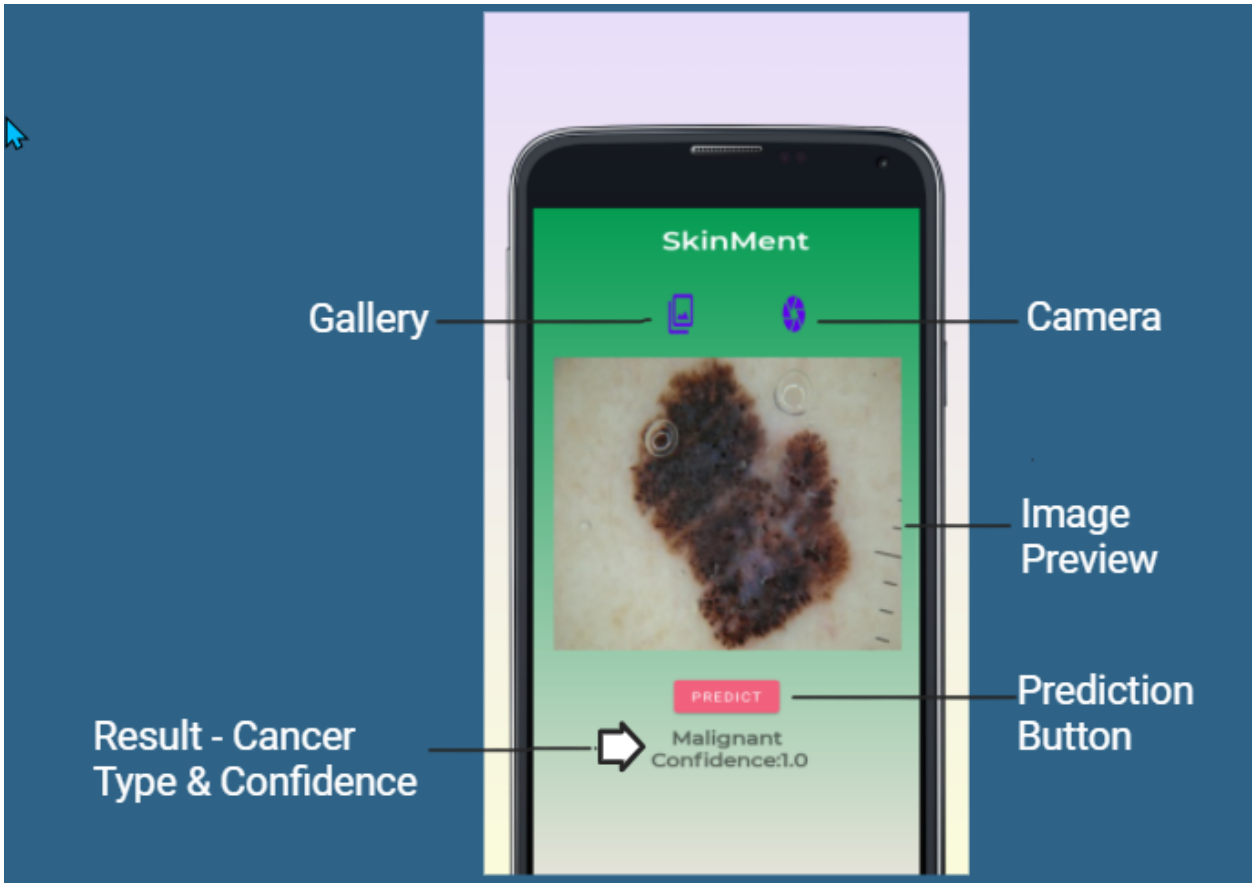
The last step is to pass the input prepared above to the interpreter and extract the output from it:

```
companion object {
        class LoaderTask(private val progressActivity:
ProgressActivity) : AsyncTask<List<Image>, Int, Unit>() {
        ...
            override fun doInBackground(vararg images:
List<Image>) {
                val imageList = images[0]

                imageList.forEachIndexed { index, image ->
```

```
                    // Read the bitmap from a local file
                    val bitmap =
BitmapFactory.decodeFile(image.file.path)
                    // Resize the bitmap so that it's
224x224
                    val resizedImage =

Bitmap.createScaledBitmap(bitmap,
progressActivity.inputImageWidth,
progressActivity.inputImageHeight, true)

                    // Convert the bitmap to a ByteBuffer
                    val modelInput =
progressActivity.convertBitmapToByteBuffer(resizedImage)

                    // Perform inference on the model

progressActivity.interpreter.run(modelInput,
progressActivity.resultArray)
                }
            }
        ...
        }
}
```

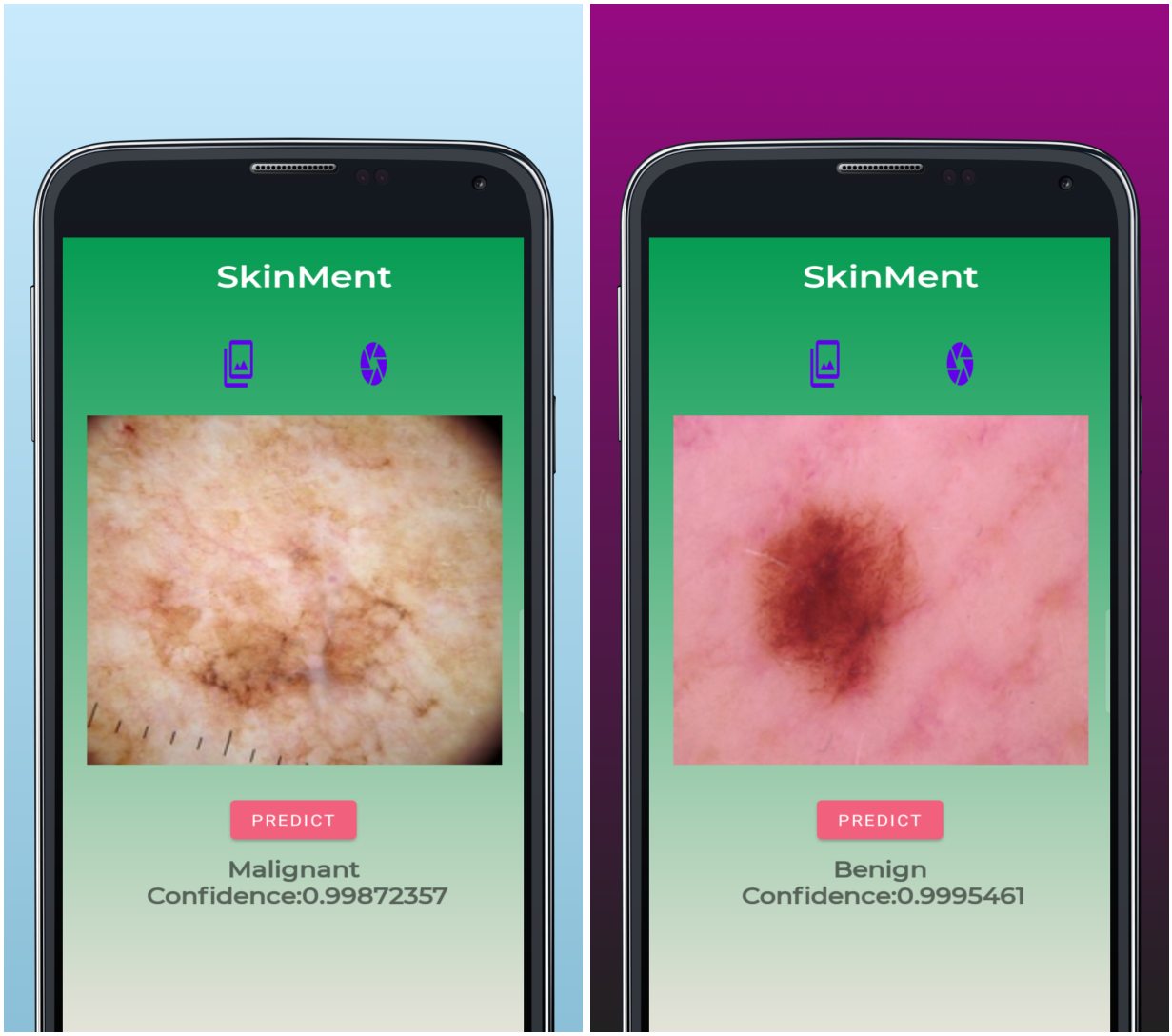
4.2 Working of Skinment application

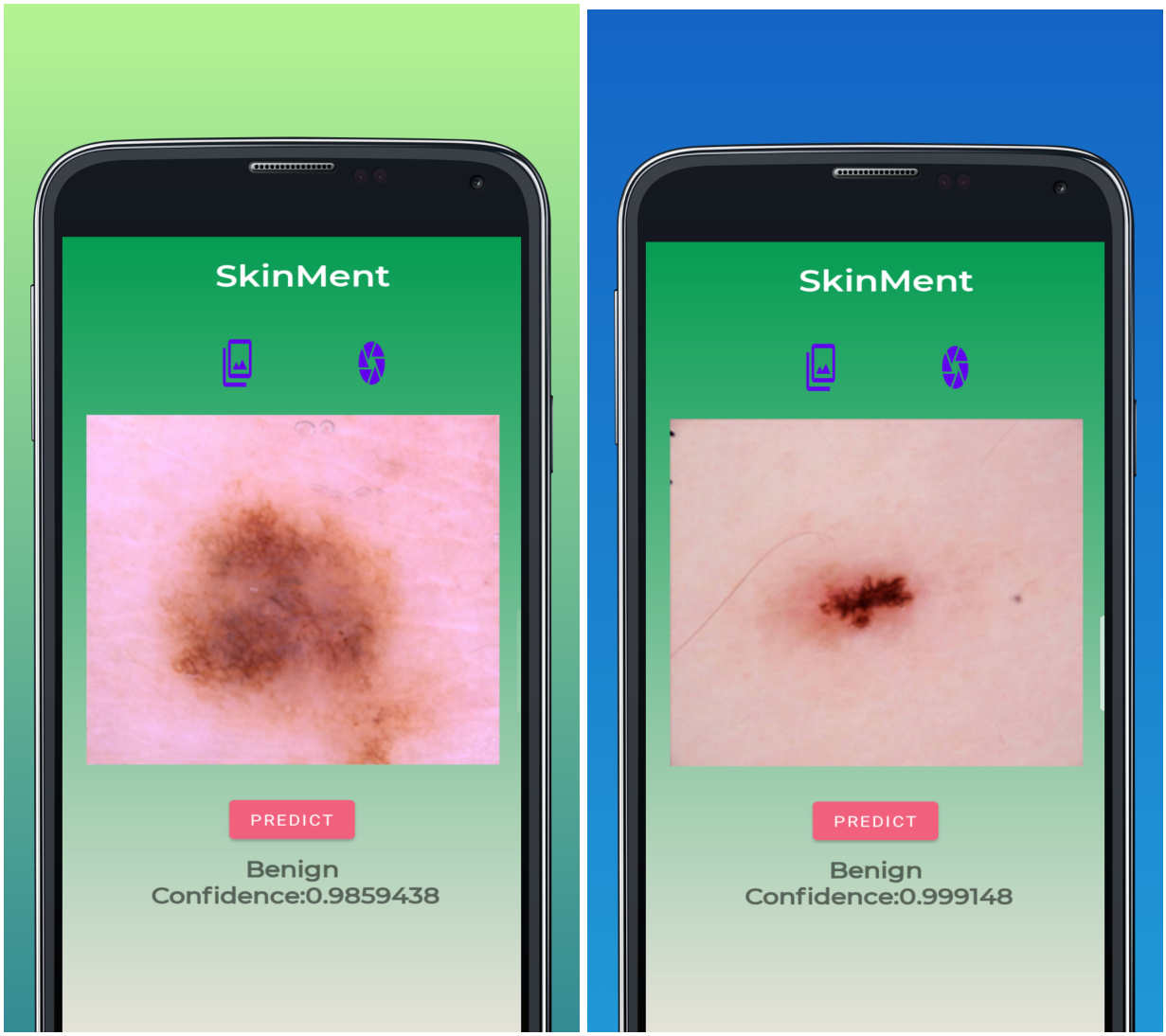
Components



Demonstration

An Android application classifying skin lesion into Malignant or Benign


SkinMent
PREDICT
Malignant
Confidence:0.99872357
SkinMent
PREDICT
Benign
Confidence:0.9995461



4.3 Features of Skinment application

- ➢ Apk Size is Less than 11 Mb
- ➢ Minimal User Interface
- ➢ No internet connection required
- ➢ Easy to use
- ➢ 77 % accurate result

4.4 Conclusion

The proposed deep CNN model could classify the melanoma types into benign class or malignant class. In this work, a less complicated model is used and the accuracy obtained was around 77 %. The application which is put forward is to a great extent an effective tool that helps in the timely as well as lively evaluation of the disease.

5. Abstract

The uncontrolled growth of abnormal epidermal cells leads to a cancer, which like any other malignancy proves to be baneful if not treated at an early stage. Prior prognosis of whichever type of skin cancer urges the possibility of betterment. But the present-day technique used to recognize cancer is a tedious process. The machine assisted approach for detection of cancer is at the same time more efficient. Deep learning is an artificial intelligence operation that emulates the working of the human brain in organizing data and designing patterns for decision making. Most modern deep learning models are based on artificial neural networks categorically convolutional neural networks. In this project we developed a deep learning architecture which focuses on the timely evaluation of skin cancer. The model could classify the melanoma into a benign or malignant class with 77%accuracy using a mobile application.

References:

https://www.upgrad.com/blog/basic-cnn-architecture/

https://www.tensorflow.org/lite/guide/android

https://heartbeat.fritz.ai/loading-and-running-a-quantized-tensorflow-lite-model-on-android-18aa7505076a

https://medium.datadriveninvestor.com/how-a-computer-looks-at-pictures-image-classification-a4992a83f46b