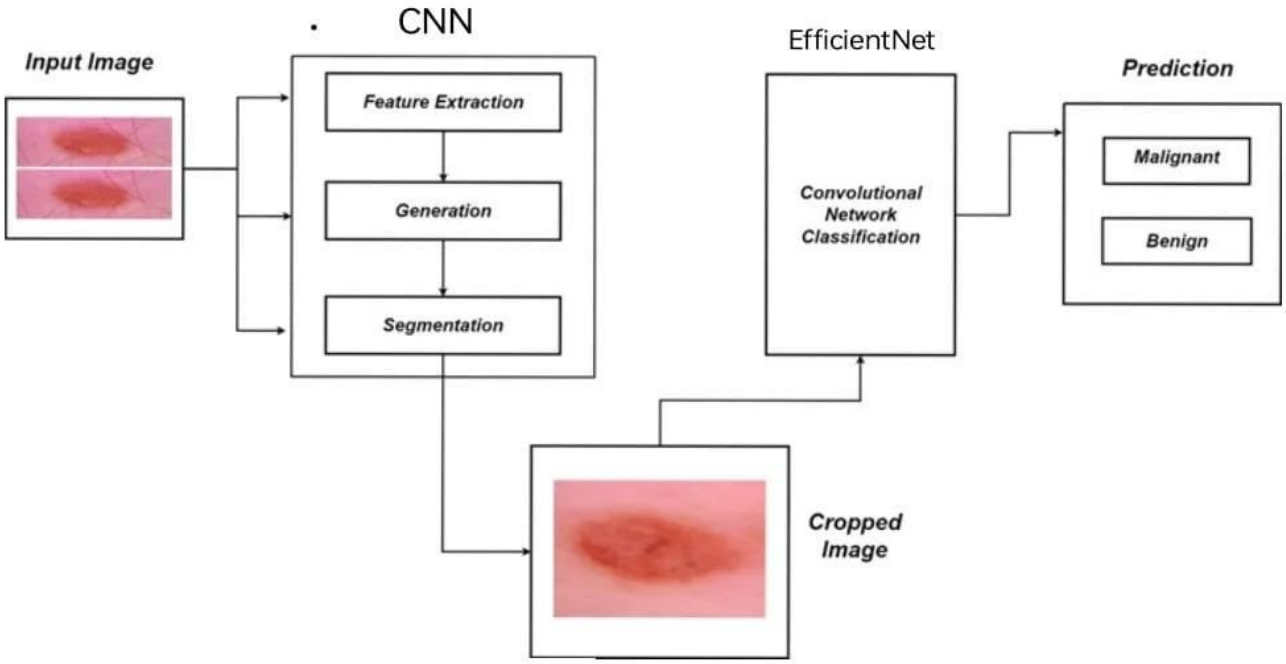
End-to-end deep learning project for detecting melanoma diseases.

Melanoma is a type of skin cancer that develops in the melanocytes, the pigment-producing cells in the skin. It is considered the most aggressive form of skin cancer and can potentially spread to other parts of the body if not detected and treated early. Cutaneous melanomas are the most prevalent type of melanoma and primarily occur on the skin, especially in sun-exposed regions. In men, melanomas frequently develop on the chest or back, while in women, the legs are more commonly affected. However, melanomas can also appear on the neck, face, or in areas not exposed to sunlight, such as the groin or beneath the fingernails. It is essential to be vigilant in monitoring these areas for any suspicious skin changes, as early detection plays a crucial role in successful melanoma treatment.

Deep learning models trained on large datasets can contribute to early detection and screening efforts for melanoma. By analysing images from individuals without apparent symptoms, these models can identify suspicious lesions that may warrant further examination, enabling earlier intervention and improved treatment outcomes. Factors include having fair skin, a history of sunburns, a family history of melanoma, a weakened immune system, and having many moles or atypical moles.This project highlights the significance of Deep Learning, specifically Convolutional Neural Networks (CNNs), in automated medical diagnosis. It emphasizes the use of CNNs for segmentation, classification, and detection of various diseases. The proposed method consists of two stages: First, the Mask and Region-based CNN is employed to automatically crop the region of interest from a dermatoscopic image. Next, a ResNet152 structure is utilized to classify lesions as either "benign" or "malignant." This approach aims to improve the accuracy and efficiency of medical diagnosis through the integration of advanced deep learning techniques.

Technical Architecture:



Project Flow:

- The user interacts with the UI to upload the file.
- Uploaded input is analyzed by the model by .
- Once the model analyzes the input the prediction is sent

Project Objectives: To detect melanoma disease with accuracy for early treatment and make the process easily accessible to everyone the novelty of the present methodology lies in its ability to perform skin lesion detection in a very quick time, which in turn helps technicians enhance their diagnostic skills. . The speed at which the methodology can accurately identify and classify skin lesions allows technicians to efficiently analyze a large number of images and make quicker and more accurate diagnostic decisions

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

The objective of this end-to-end deep learning project is to develop a and accurate system for the early detection of melanoma, a deadly form of skin cancer. Melanoma is known for its rapid progression, and early diagnosis is critical for effective treatment. Current methods for detecting melanoma involve visual inspection by dermatologists, which can be time-consuming and subject to human error therefore project aims to create an automated system that can assist medical professionals in diagnosing melanoma more quickly and accurately.The proposed model simplifies the classification process by directly using raw images as input. It automatically learns important features from the images, removing the need for complicated lesion segmentation and feature extraction steps. This approach streamlines the classification process and improves efficiency

Activity 2: Business requirements

Here are some potential business requirements for an ecommerce product delivery estimation predictor using deep learning:

Accurate prediction: The predictor must be able to accurately predict the type of skin cancer. The accuracy of the prediction is crucial for doctors, citizens, and other stakeholders to make informed decisions on the production.

User-friendly interface: The predictor must have a user-friendly interface that is easy to navigate and understand. The interface should present the results of the predictor in a clear and concise manner to enable doctors and other stakeholders to make informed decisions.

Scalability: The predictor must be able to scale up based on the prediction from our product. The model should be able to handle any size of data without compromising on its accuracy or efficiency.

Activity 3: Literature Survey

Jaleel, J.A, Salim, S they proposed an automated melanoma skin cancer diagnostic based on backpropagation ANN. In this model, a feed forward multilayer is used and backpropagation algorithm for the training since there will be a desired output, for which the training is done.
The system utilized a 2D-wavelet transform technique for extracting features. The proposed methodology involves using a multilayer feed forward network to analyse five features. The network consists of one hidden layer with two hidden neurons and an output layer with one output neuron. The activation function used is a linear function, which produces an output of either 0 or 1

Bisla introduced a deep learning approach that involved data purification and data augmentation using a Generative Adversarial Network (GAN). Their proposed system utilized decoupled deep convolutional GANs to generate additional data. They refined a pre-trained ResNet-50 model with the purified and augmented dataset and employed it to classify dermoscopic images into three categories: melanoma, SK, and nevus

Harangi et al [15] The authors developed an ensemble model specifically for classifying three classes of skin cancer. They used four well-known CNN architectures, namely GoogleNet, AlexNet, ResNet, and VGGNet, to build individual models. Each of these models was trained on their dataset and tested for accuracy. The results showed that each CNN model achieved the following accuracies for skin cancer classification:

- GoogleNet: 84.2%
- AlexNet: 84.8%
- ResNet: 82.8%
- VGGNet: 81.4%

However, the authors observed that by combining the predictions of three specific models, namely GoogleNet, AlexNet, and VGGNet, they were able to create an ensemble model that further improved the accuracy. The accuracy achieved by this ensemble model was 83.8%.

To improve the performance of their ensemble method, the applied data augmentation and color normalization techniques on the ISIC 2018 dataset. Data augmentation involves generating additional training samples by applying various transformations to the original images, such as rotations, flips, and scaling. This process helps to increase the diversity of the training data, leading to a more robust and generalizable model. Color normalization is used to standardize the color distribution across the dataset, ensuring that differences in lighting conditions or color variations do not impact the classification results.

Activity 4: Social or Business Impact.

The model's predictions can assist in early diagnosis, offering the potential for better patient outcomes and personalized care plans, ultimately improving the disease management. Cost savings in healthcare, better access to services, and hope for effective treatments contribute to overall satisfaction, making early detection a crucial element in care. By leveraging deep learning algorithms, researchers and healthcare professionals can gain deeper insights into the disease, enhance early detection, and develop more effective therapeutic interventions.

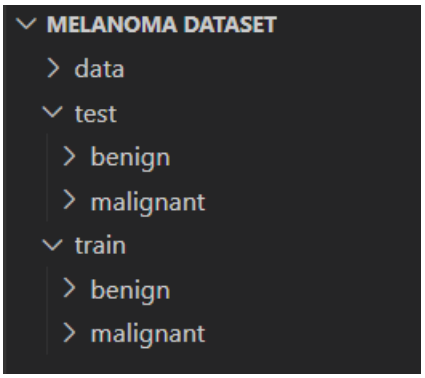
Creating a business model or revenue strategy for melanoma disease detection entails providing a valuable service or product associated with the identification, diagnosis, or care of this condition. This can include diagnostic services, data analysis, software licenses, educational programs, and more, depending on the specific offerings and target marke

Milestone 2: Data Collection

There are 2 types of melanoma diseases

1. benign
2. malignant

For building a Deep learning model, we need to divide the dataset into test and train



Dataset link: https://www.kaggle.com/datasets/fanconic/skin-cancer-malignant-vs-benign

Milestone 3: Image Preprocessing

In this milestone the collected dataset suffers from imbalanced classes, meaning some classes have significantly fewer samples than others. Hence the dataset undergoes several pre-processing steps, including normalization, standardization, resizing, de noising, and format conversion, skull removal, tissue segmentation etc.

Activity 1: Import the ImageDataGenerator library

Image data augmentation in deep learning involves modifying training images with operations like rotation, flipping, scaling, and brightness adjustments to create additional training samples, improving model generalization.
This technique helps prevent overfitting and enhances the model's ability to recognize objects in various orientations and conditions. Popular libraries like TensorFlow and Keras provide tools for implementing image data augmentation.
The Keras deep learning neural network library offers the functionality to train models with image data augmentation using the ImageDataGenerator class.

Let us import the ImageDataGenerator class from tensorflow Keras:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
```

Activity 2: Configure ImageDataGenerator class

The ImageDataGenerator class is initialized, and its settings for various data augmentation techniques are configured.
There are five primary data augmentation techniques for image data, which can be specified using the following arguments:

- For shifting the image horizontally and vertically, you can use the width_shift_range and height_shift_range arguments.
- To apply horizontal and vertical flips to the images, you can use the

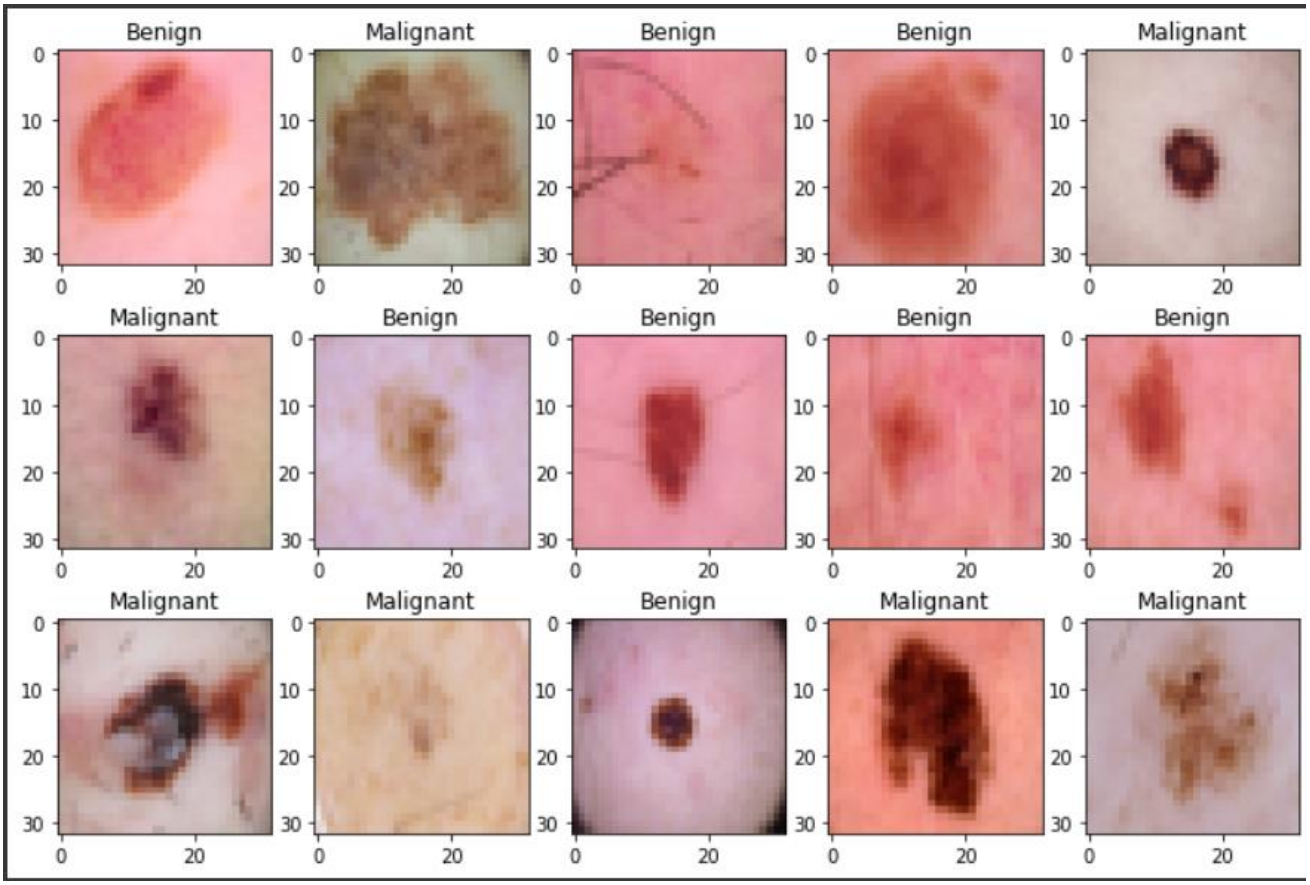
- Image rotations can be introduced with the rotation_range argument. Adjusting image brightness can be achieved by setting the brightness_range argument.
- To zoom in or out of the images, you can specify the zoom_range.

An instance of the ImageDataGenerator class can be created for both training and testing purposes.

Activity 3 : Apply ImageDataGenerator functionality to Trainset and Te set

```
from PIL import Image
#load the training and testing files
directory_benign_train = '/content/train/benign'
directory_malignant_train = '/content/train/malignant'
directory_benign_test = '/content/test/benign'
directory_malignant_test = '/content/test/malignant'
read = lambda imname: np.asarray(Image.open(imname).convert('RGB').resize((32,32)))

#reading files
img_benign_train = [read(os.path.join(directory_benign_train, filename)) for filename in os.listdir(directory_benign_train)]
img_malignant_train = [read(os.path.join(directory_malignant_train, filename)) for filename in os.listdir(directory_malignant_train)]


img_benign_test = [read(os.path.join(directory_benign_test, filename)) for filename in os.listdir(directory_benign_test)]
img_malignant_test = [read(os.path.join(directory_malignant_test, filename)) for filename in os.listdir(directory_malignant_test)]

type(img_benign_train)
```

Activity 4 : Visualizing the sample data



Milestone 4: Model Building 1

ResNet-50 consists of 50 layers, including convolutional layers, pooling layers, and fully connected layers. It's a pre-defined architecture with skip connections (residual connections) that helps mitigate the vanishing gradient problem and allows you to train very deep networks.

Activity 1: Importing the Model Building Libraries

Importing the required libraries for our project

```
#importing libraries
#numpy for arrays
import numpy as np
#pandas for data analaysis
import pandas as pd
#os for directories
import os
#matplot for displaying images and plotting images
import matplotlib.pyplot as plt
import seaborn as sns
#glob is used for files and determing path
from glob import glob
#random seed to generate random values. Sending images to training
np.random.seed(21)
```

Activity 2: Adding CNN Layers

```
#applying CNN model
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import Adam, RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator

def build_cnn_model(input_shape = (32, 32, 3), num_classes=2):
    model = Sequential()

    model.add(Conv2D(64, kernel_size=(3,3), padding='Same', input_shape = input_shape, activation='relu', kernel_initializer = 'glorot_uniform'))
    model.add(MaxPool2D(pool_size = (2,2)))
    model.add(Dropout(0.25))


    model.add(Conv2D(64, kernel_size=(3,3), padding='Same', activation='relu', kernel_initializer = 'glorot_uniform'))
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(Dropout(0.25))


    model.add(Flatten())

    model.add(Dense(128, activation='relu', kernel_initializer='normal'))
    model.add(Dense(128, activation='relu', kernel_initializer='normal'))

    model.add(Dense(num_classes, activation = 'softmax'))
    model.summary()
    optimizer= Adam(lr=0.001)

    model.compile(optimizer = optimizer, loss='binary_crossentropy', metrics=["accuracy"])

    return model
```

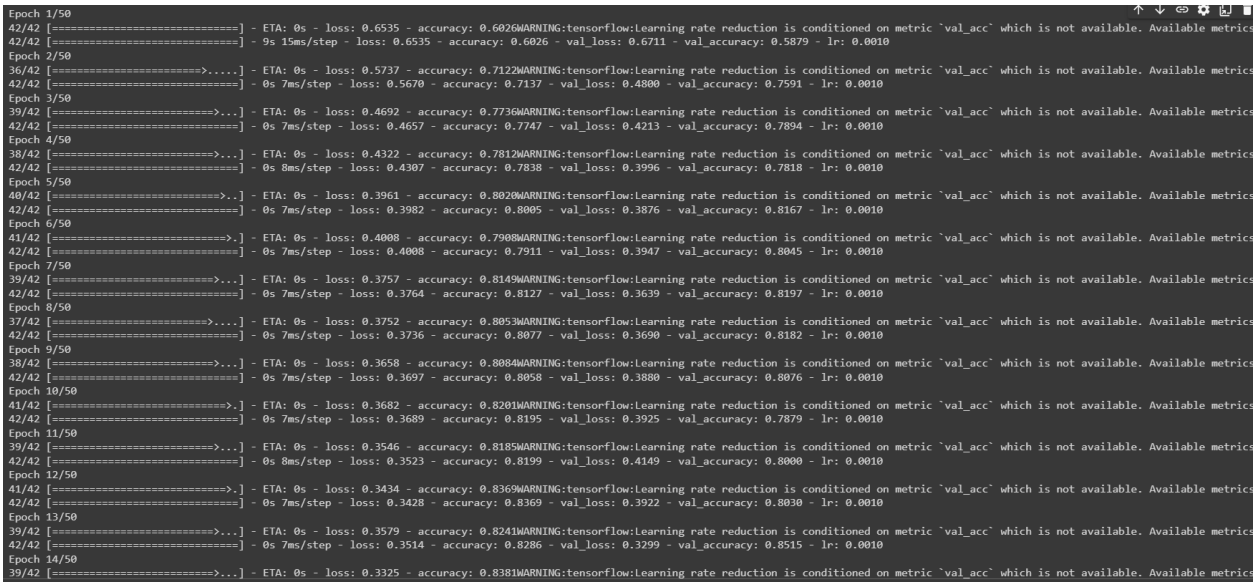
Activity 3 : summary of the model and its layers

```
model_cnn = build_cnn_model()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 dropout (Dropout)           (None, 16, 16, 64)        0

 conv2d_1 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 128)               524416

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 2)                 258

=================================================================
Total params: 579,906
Trainable params: 579,906
Non-trainable params: 0
_________________________________________________________________
/usr/local/lib/python3.8/dist-packages/keras/optimizers/optimizer_v2/adam.py:110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

Activity 4: Train The model

```
from keras.callbacks import ReduceLROnPlateau
learning_rate_annealer = ReduceLROnPlateau(monitor='val_acc',mode='max',
                                           patience=3,
                                           verbose=1,
                                           factor=0.5,
                                           min_lr = 1e-4)


history = model_cnn.fit(X_train,
                        y_train,
                        validation_split=0.2,
                        epochs=50,
                        batch_size = 64,
                        verbose=1,
                        callbacks=[learning_rate_annealer],validation_data=(X_test,y_test))



print(history.history.keys())
```



Model Building 2

Activity 1: Adding CNN Layers

```
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import Adam, RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
def build_cnn_model(input_shape = (32, 32, 3), num_classes=2):
    model = Sequential()
    model.add(base_model)
    model.add(Flatten())

    model.add(Dense(128, activation='relu', kernel_initializer='normal'))
    model.add(Dense(128, activation='relu', kernel_initializer='normal'))

    model.add(Dense(2, activation = 'softmax'))
    model.summary()
    optimizer= Adam(lr=0.001)

    model.compile(optimizer = optimizer, loss='binary_crossentropy', metrics=["accuracy"])

    return model
```

Activity 2 :summary of the model and its layers

```
model_cnn = build_cnn_model()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 model (Functional)          (None, 1, 1, 2048)        28513520

 flatten_1 (Flatten)         (None, 2048)              0

 dense_3 (Dense)             (None, 128)               262272

 dense_4 (Dense)             (None, 128)               16512

 dense_5 (Dense)             (None, 2)                 258

=================================================================
Total params: 28,792,562
Trainable params: 28,619,826
Non-trainable params: 172,736
_________________________________________________________________
```

Activity 3: Train The model

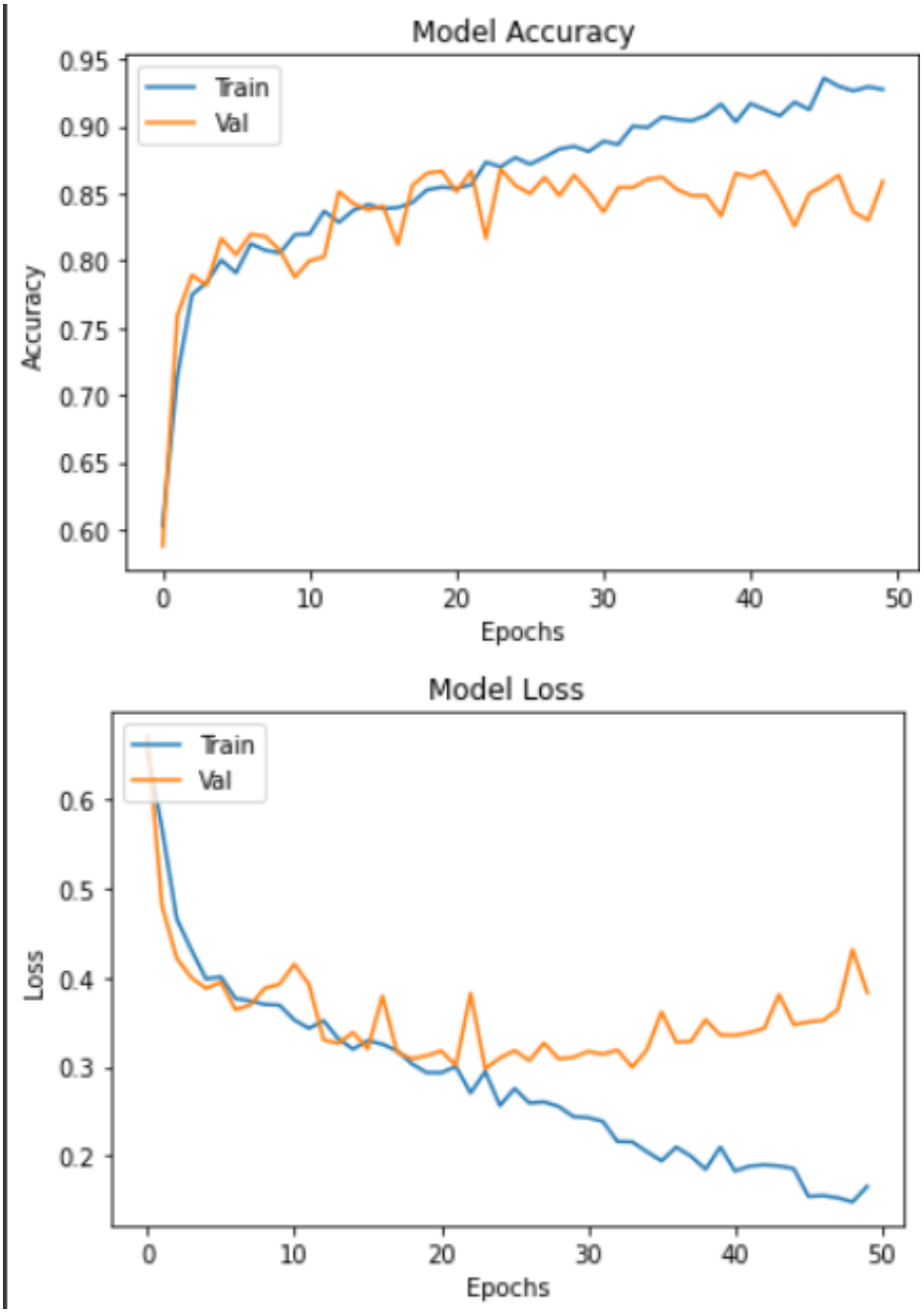
```
from keras.callbacks import ReduceLROnPlateau
learning_rate_annealer = ReduceLROnPlateau(monitor='val_acc',mode='max',
                                           patience=3,
                                           verbose=1,
                                           factor=0.5,
                                           min_lr = 1e-4)


history = model_cnn .fit(X_train,
                    y_train,
                    validation_split=0.2,
                    epochs=50,
                    batch_size = 64,
                    verbose=1,
                    callbacks=[learning_rate_annealer],validation_data=(X_test,y_test))



print(history.history.keys())
```

```
Epoch 1/50
42/42 [==============================] - ETA: 0s - loss: 0.4469 - accuracy: 0.7812WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 38s 222ms/step - loss: 0.4469 - accuracy: 0.7812 - val_loss: 6.8728 - val_accuracy: 0.5455 - lr: 0.0010
Epoch 2/50
42/42 [==============================] - ETA: 0s - loss: 0.3152 - accuracy: 0.8665WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 4s 91ms/step - loss: 0.3152 - accuracy: 0.8665 - val_loss: 1.0110 - val_accuracy: 0.5455 - lr: 0.0010
Epoch 3/50
42/42 [==============================] - ETA: 0s - loss: 0.2903 - accuracy: 0.8805WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 4s 91ms/step - loss: 0.2903 - accuracy: 0.8805 - val_loss: 3.8999 - val_accuracy: 0.5455 - lr: 0.0010
Epoch 4/50
42/42 [==============================] - ETA: 0s - loss: 0.1994 - accuracy: 0.9188WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 4s 92ms/step - loss: 0.1994 - accuracy: 0.9188 - val_loss: 1.2811 - val_accuracy: 0.6485 - lr: 0.0010
Epoch 5/50
42/42 [==============================] - ETA: 0s - loss: 0.1532 - accuracy: 0.9344WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 4s 103ms/step - loss: 0.1532 - accuracy: 0.9344 - val_loss: 1.1890 - val_accuracy: 0.7697 - lr: 0.0010
Epoch 6/50
42/42 [==============================] - ETA: 0s - loss: 0.1750 - accuracy: 0.9287WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 4s 91ms/step - loss: 0.1750 - accuracy: 0.9287 - val_loss: 0.9783 - val_accuracy: 0.7697 - lr: 0.0010
Epoch 7/50
42/42 [==============================] - ETA: 0s - loss: 0.1522 - accuracy: 0.9401WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 4s 93ms/step - loss: 0.1522 - accuracy: 0.9401 - val_loss: 0.7786 - val_accuracy: 0.8106 - lr: 0.0010
Epoch 8/50
42/42 [==============================] - ETA: 0s - loss: 0.1232 - accuracy: 0.9541WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 4s 106ms/step - loss: 0.1232 - accuracy: 0.9541 - val_loss: 0.6177 - val_accuracy: 0.8197 - lr: 0.0010
Epoch 9/50
42/42 [==============================] - ETA: 0s - loss: 0.1049 - accuracy: 0.9628WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 4s 93ms/step - loss: 0.1049 - accuracy: 0.9628 - val_loss: 0.5810 - val_accuracy: 0.8348 - lr: 0.0010
Epoch 10/50
42/42 [==============================] - ETA: 0s - loss: 0.0692 - accuracy: 0.9757WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 4s 92ms/step - loss: 0.0692 - accuracy: 0.9757 - val_loss: 0.5703 - val_accuracy: 0.8212 - lr: 0.0010
Epoch 11/50
42/42 [==============================] - ETA: 0s - loss: 0.1286 - accuracy: 0.9537WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 4s 92ms/step - loss: 0.1286 - accuracy: 0.9537 - val_loss: 0.4433 - val_accuracy: 0.8530 - lr: 0.0010
Epoch 12/50
42/42 [==============================] - ETA: 0s - loss: 0.0760 - accuracy: 0.9738WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 4s 91ms/step - loss: 0.0760 - accuracy: 0.9738 - val_loss: 0.4763 - val_accuracy: 0.8455 - lr: 0.0010
Epoch 13/50
42/42 [==============================] - ETA: 0s - loss: 0.0620 - accuracy: 0.9765WARNING:tensorflow:Learning rate reduction is conditioned on metric `val_acc` which is not available. Available metrics a
42/42 [==============================] - 4s 91ms/step - loss: 0.0620 - accuracy: 0.9765 - val_loss: 0.7439 - val_accuracy: 0.8667 - lr: 0.0010
```

Milestone 5:Deployment

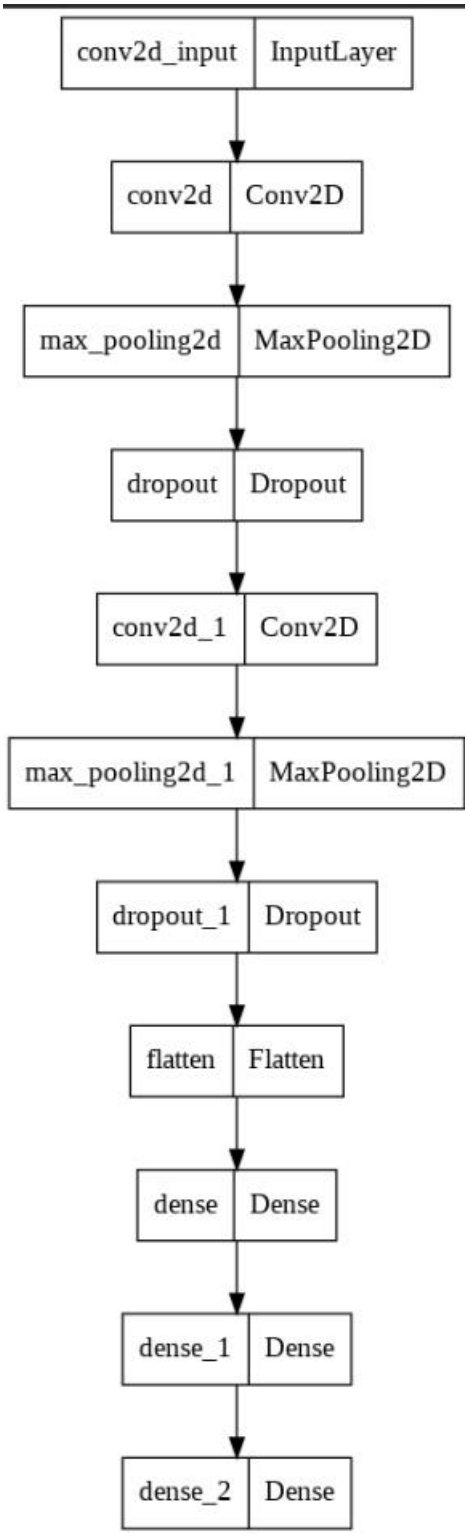
Ploting the accuracy of the model

```
#plotting acuracy and loss
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')

plt.legend(['Train', 'Val'], loc='upper left')
plt.show()


plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')

plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```


Model Accuracy
Train
Val
Accuracy
Epochs
Model Loss
Train
Val
Loss
Epochs

Tensor flow of the model

```
import tensorflow as tf
tf.keras.utils.plot_model(model_cnn)
```

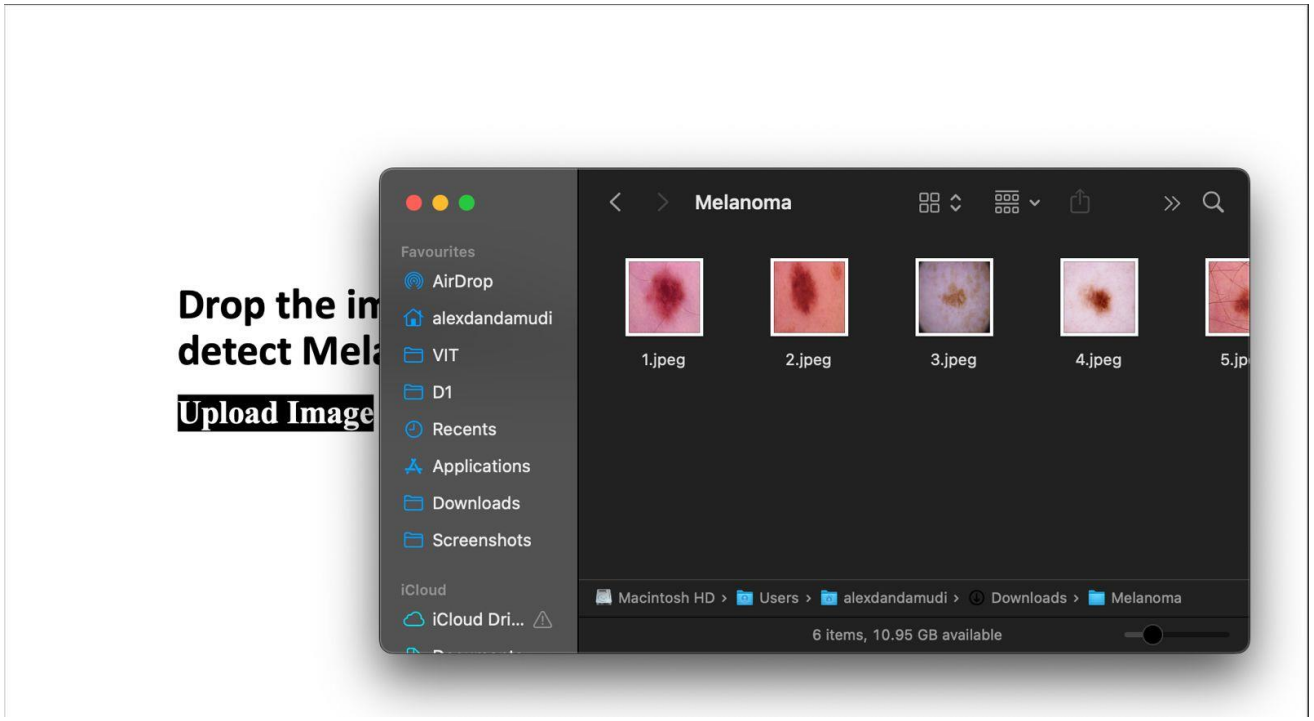


Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built.
We will be using the streamlit package for our website development.
Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps.

The HTML Pages

When you click on the upload image, the files box appears choose the image to upload



Build python code

Task 1: Importing Libraries

The first step is usually importing the libraries that will be needed in the program:

```
import graphlib
from future import division, print_function

import sys
import os
import glob
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.applications.imagenet.utils import preprocess_input, decode_predictions
from tensorflow.keras.models import load_model
from tensorflow.keras import backend
from tensorflow.keras import backend
from tensorflow import keras
import tensorflow as tf

from skimage.transform import resize

from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer
```

Importing the flask module in the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument Pickle library to load the model file.

Creating our flask application and loading our model by using load_model method

```
#Define a flask app
app = Flask(__name__)
#Load the trained model
model=load_model('moderatedementia.h5')
```

Routing to the html Page

Here, the previously produced HTML page is accessed using the defined constructor.

The index.html function is connected to the '/' URL in the example above. Thus, the HTML page will be rendered when a web server's home page is viewed in a browser. The POST or GET methods can be used to access an image when browsing an HTML page.

```
#Define a flask app
app = Flask(__name__)
#Load the trained model
model=load_model('moderatedementia.h5')

@app.route('/', methods=['GET'])
def index():
    #Main_Page
    return render_template('index.html')
@app.route('/Image',methods=['POST','GET'])
def prediction():
    #It will direct you to the prediction page
    return render_template('base.html')
```

Showcasing prediction on UI:

Here we are defining a function which requests the browsed file from the html page using the post method. The requested picture file is then saved to the uploads folder in this same directory using OS library.

Using the load image class from Keras library we are retrieving the saved picture from the path declared. We are applying some image processing techniques and then sending that preprocessed image to the model for predicting the class. This returns the numericalvalue of a class (like 0,1 ,2 etc.) which lies in the 0th index of the variable.

This numericalvalue is passed to the index variable declared. This returns the name of the class. This name is rendered to the predict variable used in the html page.

```python
@app.route('/', methods=['GET'])
def upload():
    if request.method == 'POST':
        f = request.files['image']
    basepath = os.path.dirname(__file__)
    file_path = os.path.join(basepath, 'uploads', secure_filename(f.filemame))
    f.save(file_path)
    img = image.load_img(file_path, target_size=(224,224))
    x = image.img_to_array(img)
    x = np.expand_dims(x,axis=0)

    with graph.as_default():
        set_session(sess)
        predicition = model.predict(x)[0][0][0]
    print(predicition)
    if predicition==0:
        text = "Mild Demented"
    elif predicition==1:
        text = "Moderate Demented"
    elif predicition==2:
        text = "Non Demented"
    else:
        text = "Very Mild Demented"
    return text
```

Predicting the results

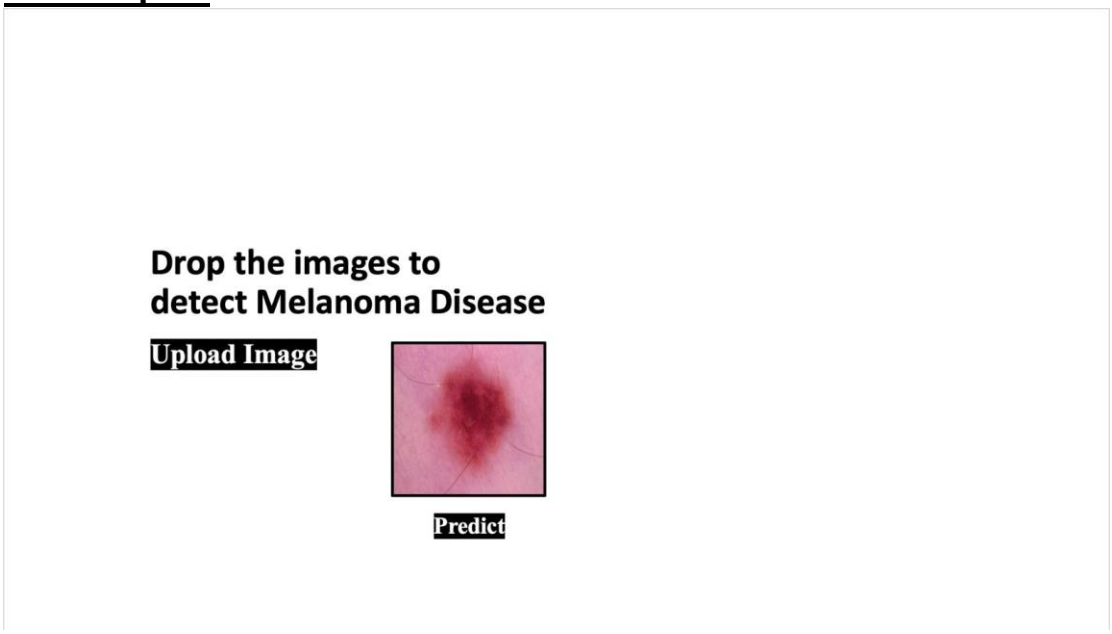
We then proceed to predict the stages of disease in the input image using model.predict function and the result is stored in the result variable.

Finally, Run the application

This is used to run the application in a local host.

```python
# running your application
if __name__ == "__main__":
    app.run()
#http://localhost:5000/ or localhost:5000
```

The output:


Drop the images to
detect Melanoma Disease
Upload Image
Predict


Drop the images to
detect Melanoma Disease
Upload Image
Found: Benign Melanoma