SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be **University** *u/s 3 of UGC Act, 1956)*

Subject code:

Course Code	18CSE388T	Course Name	ARTIFICIAL NEURAL NETWORKS

Department of Computational Intelligence
School of Computing
SRMIST
Kattankulathur
ODD Semester: 2022-2023

TITLE OF THE PROJECT: **SKIN CANCER DETECTION**

STUDENT DETAILS:

- SAYAK DAS (RA2011026010101)
- ROOPAL SOOD (RA2011026010103)
- VIABHAV SHARMA (RA2011026010106)

Submitted to : (Dr. A. JACKULIN MAHARIBA)

PROBLEM STATEMENT

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

There are mainly 4 types of skin cancers detected as of now –

- Basal cell Carcinoma
- squamous cell carcinoma
- Melanoma
- actinic keratosis

We have to detect skin cancer patches specially Melanoma as it can be more deadly if not detected early. Multilayer Convolutional Neural Network is used for the model architecture. The outcome of this project is to design a diagnostic model that could effectively and efficiently detect the type of skin cancer.

INTRODUCTION

The technique/Architecture we used to solve the problem is Convolutional Neural Network(CNN) .

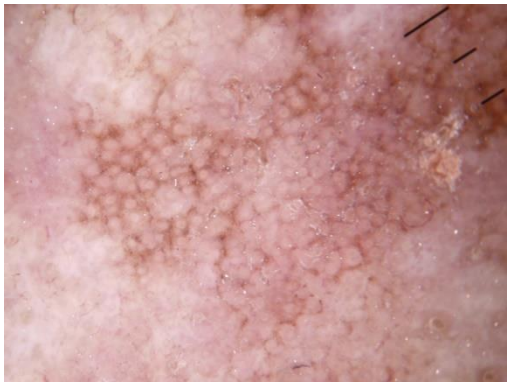
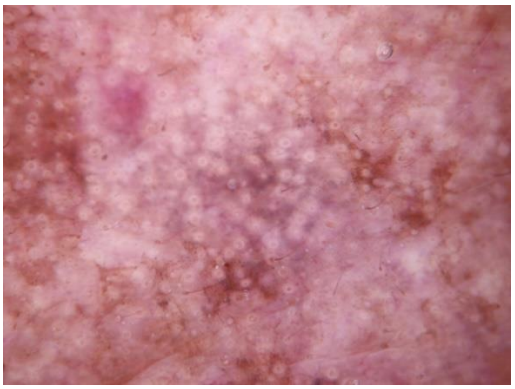
Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

Dataset:

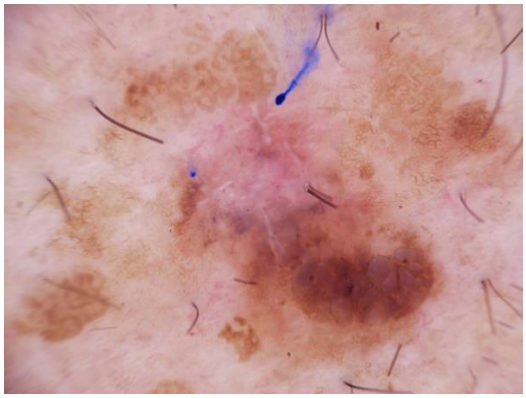
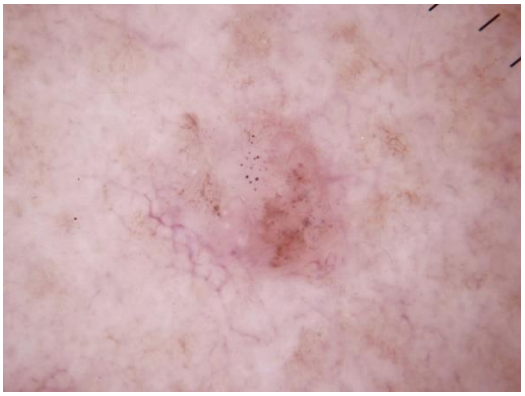
There are two folders in the dataset one is for training purpose and one is for testing purpose.

This dataset consists of total 2357 images , which are further classified into 9 different categories of skin diseases/patches.

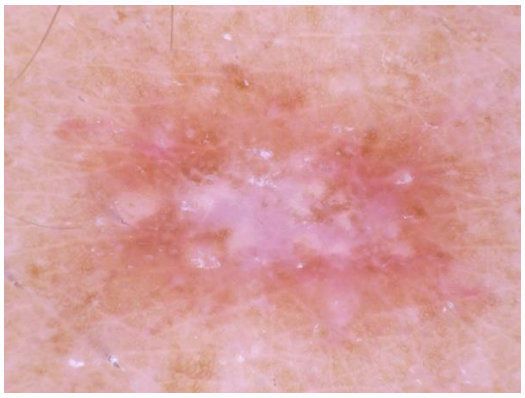
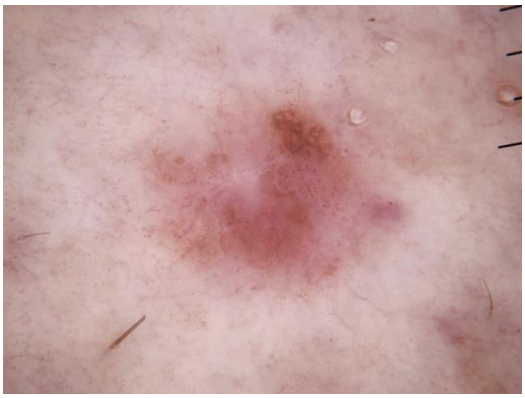
- Actinic keratosis:

- Basal cell carcinoma:

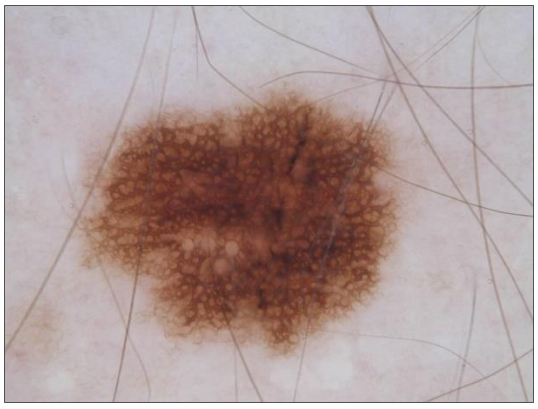



- Dermatofibroma:

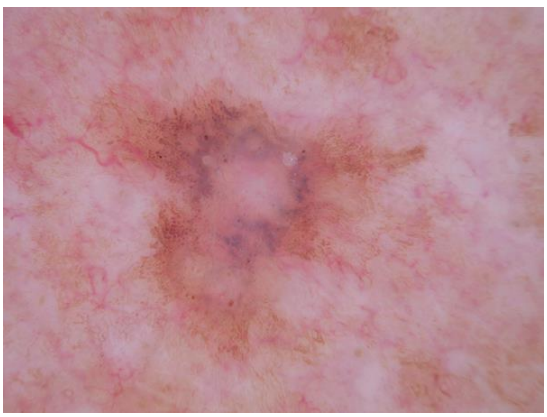



- Melanoma:

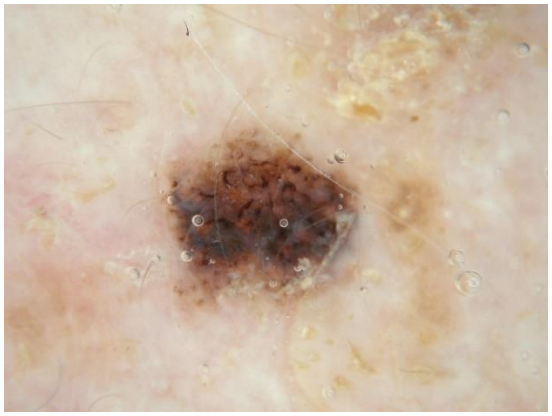
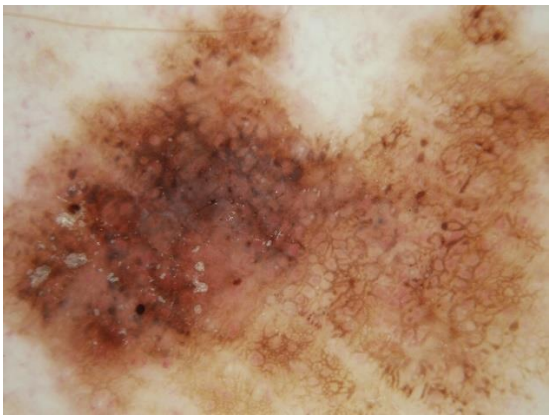

- Nevus:



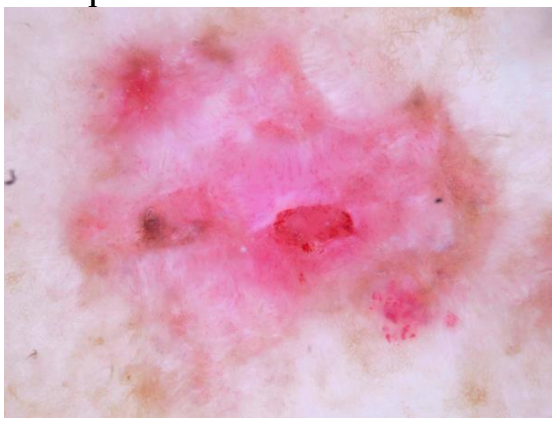
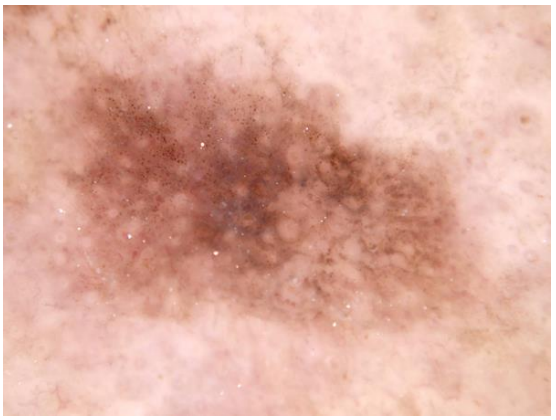

- Pigmented benign keratosis:

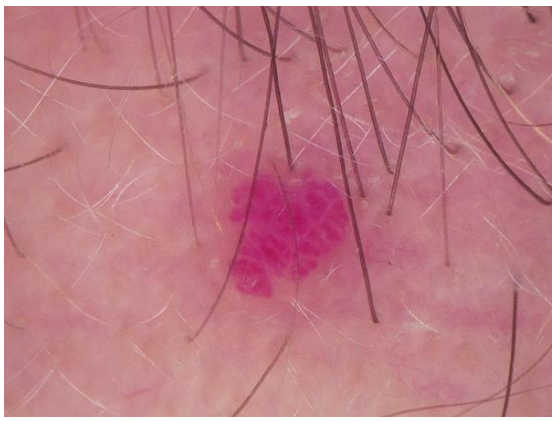
- Seborrheic keratosis:

- Squamous cell carcinoma:

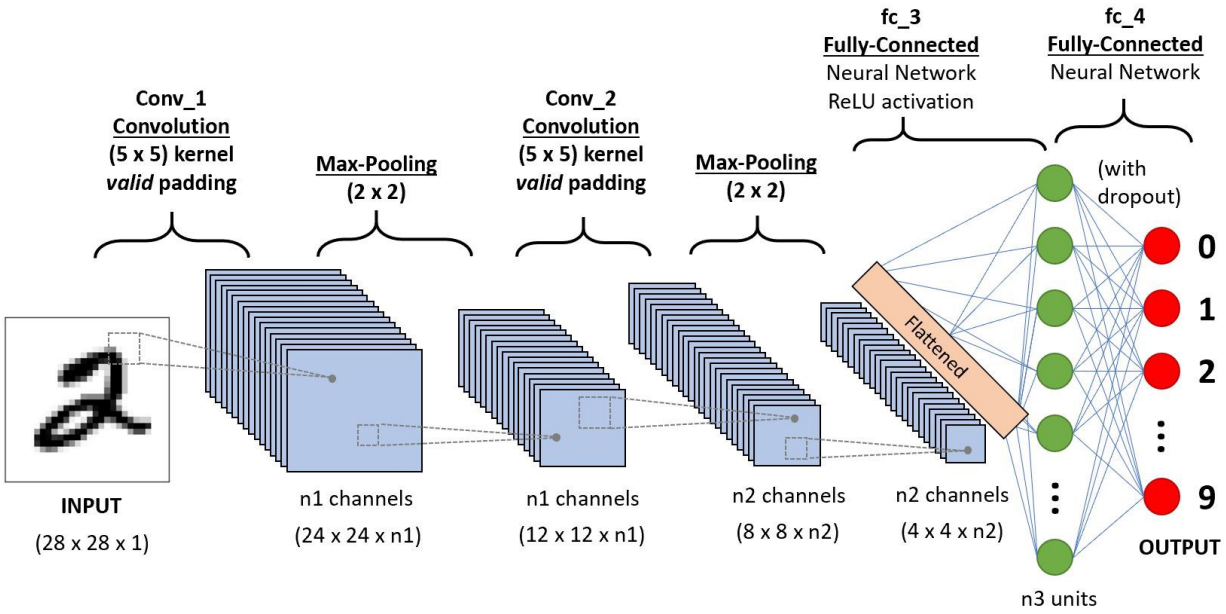
- vascular lesion:




Training Dataset : 2239 images ; **after Augmentation**:8992 images
Testing Dataset: 118 images
Validation Dataset: 2247 images (after augmentation)

Source: **https://www.kaggle.com/datasets/nodoubttome/skin-cancer9-classesisic**

METHODOLOGY



Convolutional Neural Network Architecture

MODEL SUMMARY:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 178, 178, 32)      896

 max_pooling2d (MaxPooling2D  (None, 89, 89, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 43, 43, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 dropout (Dropout)           (None, 20, 20, 128)       0

 flatten (Flatten)           (None, 51200)             0

 dense (Dense)               (None, 128)               6553728

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 9)                 1161

=================================================================
Total params: 6,648,137
Trainable params: 6,648,137
Non-trainable params: 0
```

DATAFLOW:

- Acquire the dataset
- Count the size of training and Testing dataset
- Visualize the dataset one instance of all the class present in the dataset.
- Visualize distribution of classes in the training dataset
- Augmentor is used for data augmentation and artificial generation of image data for machine learning tasks to get more accurate and precise result. It is primarily a data augmentation tool.
- Autotune the training and validation dataset

THE LAYERS USED IN NEURAL NETWORK

- **Input layer** - The attributes of the input dataset contain values that were falling within different ranges. To make the CNN training more efficient and obtain high accuracy, we normalized and pre-processed the dataset.
- **Hidden layers** - Seven hidden layers which uses ReLU activation function.
- **Output layer** – Output layer uses softmax activation function. There are 9 output classes (neurons) in the output layer which classify 9 types of result.

IMPLEMENTATION AND RESULT

Import all required libraries:

```
import pathlib
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping
from tensorflow.keras.preprocessing.image import load_img
```

Load the Dataset:

```
data_dir_train = pathlib.Path("C:\\Users\\dassa\\Downloads\\Skin cancer ISIC The
International Skin Imaging Collaboration\\Train\\")
data_dir_test = pathlib.Path("C:\\Users\\dassa\\Downloads\\Skin cancer ISIC The
International Skin Imaging Collaboration\\Test\\")

#Train Image count
image_count_train = len(list(data_dir_train.glob('*/*.jpg')))
print(image_count_train)

#Test Image count
image_count_test = len(list(data_dir_test.glob('*/*.jpg')))
print(image_count_test)
```

2239
118

Visualize one instance of all the class present in the dataset:

```
#Visualize one instance of all the class present in the dataset.

#image_dataset_from_directory() will return a tf.data.Dataset that yields
batches of images from the subdirectories.
#label_mode is categorial, the labels are a float32 tensor of shape (batch_size,
num_classes), representing a one-hot encoding of the class index.
image_dataset =
tf.keras.preprocessing.image_dataset_from_directory(data_dir_train,batch_size=32
,image_size=(180,180),
                                                    label_mode='
categorical',seed=123)

#all the classes of Skin Cancer
class_names = image_dataset.class_names
```

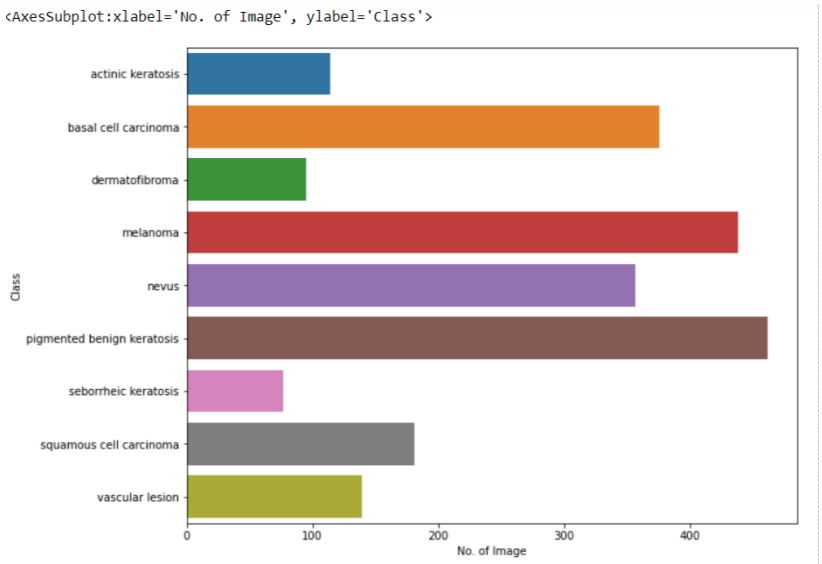
```
#Dictionary to store the path of image as per the class
files_path_dict = {}

for c in class_names:
    files_path_dict[c] = list(map(lambda
x:str(data_dir_train)+'/'+c+'/'+x,os.listdir(str(data_dir_train)+'/'+c)))

#Visualize image
plt.figure(figsize=(15,15))
index = 0
for c in class_names:
    path_list = files_path_dict[c][:1]
    index += 1
    plt.subplot(3,3,index)
    plt.imshow(load_img(path_list[0],target_size=(180,180)))
    plt.title(c)
    plt.axis("off")
```

Visualize the training dataset:

```
def class_distribution_count(directory):

    #count number of image in each classes
    count= []
    for path in pathlib.Path(directory).iterdir():
        if path.is_dir():
            count.append(len([name for name in os.listdir(path)
                              if os.path.isfile(os.path.join(path, name))]))

    #name of the classes
    sub_directory = [name for name in os.listdir(directory)
                     if os.path.isdir(os.path.join(directory, name))]

    #return dataframe with image count and class.
    return pd.DataFrame(list(zip(sub_directory,count)),columns =['Class', 'No.
of Image'])

df = class_distribution_count(data_dir_train)
df
```

```
#Visualize the Number of image in each class.
import seaborn as sns
plt.figure(figsize=(10, 8))
sns.barplot(x="No. of Image", y="Class", data=df,
            label="Class")
```

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



Augmentation:

```
path_to_training_dataset="C:\\Users\\dassa\\Downloads\\Skin cancer ISIC The
International Skin Imaging Collaboration\\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
```

```
Initialised with 114 image(s) found.
Output directory set to C:\Users\dassa\Downloads\Skin cancer ISIC The International Skin Imaging Collaboration\Train\actinic ke
ratosis\output.
Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=600x450 at 0x15C51B35310>: 100%|████████| 500/500 [00:04<0
0:00, 120.31 Samples/s]
Initialised with 376 image(s) found.
Output directory set to C:\Users\dassa\Downloads\Skin cancer ISIC The International Skin Imaging Collaboration\Train\basal cell
carcinoma\output.
Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=600x450 at 0x15C4FE82F70>: 100%|████████| 500/500 [00:04<0
0:00, 121.97 Samples/s]
Initialised with 95 image(s) found.
Output directory set to C:\Users\dassa\Downloads\Skin cancer ISIC The International Skin Imaging Collaboration\Train\dermatofib
roma\output.
Processing <PIL.Image.Image image mode=RGB size=600x450 at 0x15C51B15E50>: 100%|████████| 500/500 [00:04<00:00, 123.12 Sample
s/s]
Initialised with 438 image(s) found.
Output directory set to C:\Users\dassa\Downloads\Skin cancer ISIC The International Skin Imaging Collaboration\Train\melanoma\o
utput.
Processing <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=1024x768 at 0x15C4FCEDA30>: 100%|████████| 500/500 [00:23<0
0:00, 20.86 Samples/s]
Initialised with 357 image(s) found.
Output directory set to C:\Users\dassa\Downloads\Skin cancer ISIC The International Skin Imaging Collaboration\Train\nevus\outp
ut.
```

Dataset size after applying Augmentor :

```
#Count total number of image generated by Augmentor.
image_count_train = len(list(data_dir_train.glob('*/output/*.jpg')))
print(image_count_train)
```

9000

```
# train dataset
train_ds = tf.keras.preprocessing.image_dataset_from_directory(data_dir_train,
batch_size=32,
                                                               image_size=(180,1
80), label_mode='categorical',
                                                               seed=123,subset="
training",
                                                               validation_split=
0.2)
```

```
Found 11239 files belonging to 9 classes.
Using 8992 files for training.
```

```
# validation dataset
val_ds
=tf.keras.preprocessing.image_dataset_from_directory(data_dir_train,batch_size=3
2,
                                                     image_size=(180,180)
, label_mode='categorical',
                                                     seed=123,subset="val
idation",
                                                     validation_split=0.2
)
```

```
Found 11239 files belonging to 9 classes.
Using 2247 files for validation.
```

Autotune:

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

CNN Model Architecture:

```
#CNN Model Architecture

#Sequential allows you to create models layer-by-layer
model = Sequential()

model.add(layers.experimental.preprocessing.Rescaling(1./255,input_shape=(180,18
0,3)))

#First Convulation layer
model.add(layers.Conv2D(32,kernel_size=(3,3),activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))

#Second Convulation Layer
model.add(layers.Conv2D(64,kernel_size=(3,3),activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))

#Third Convulation Layer
model.add(layers.Conv2D(128,kernel_size=(3,3),activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2,2)))

#Dropout layer with 50% Fraction of the input units to drop.
model.add(layers.Dropout(0.5))

#Flatten Layer
##Keras.layers.flatten function flattens the multi-dimensional input tensors
into a single dimension.
model.add(layers.Flatten())

#Dense Layer
model.add(layers.Dense(128,activation='relu'))

#Dropout layer with 25% Fraction of the input units to drop.
```

```
model.add(layers.Dropout(0.25))

#Dense Layer with softmax activation function.
#Softmax is an activation function that scales numbers/logics into
probabilities.
model.add(layers.Dense(len(class_names),activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 178, 178, 32)      896

 max_pooling2d (MaxPooling2D  (None, 89, 89, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 43, 43, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 dropout (Dropout)           (None, 20, 20, 128)       0

 flatten (Flatten)           (None, 51200)             0

 dense (Dense)               (None, 128)               6553728

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 9)                 1161

=================================================================
Total params: 6,648,137
Trainable params: 6,648,137
Non-trainable params: 0
```

Compile the Model:

```
#Compile the Model

#Adam optimization: is a stochastic gradient descent method that is based on
adaptive estimation of first-order and second-order moments.
#categorical_crossentropy: Used as a loss function for multi-class
classification model where there are two or more output labels.

model.compile(optimizer="Adam",loss="categorical_crossentropy",metrics=["accurac
y"])

#ModelCheckpoint callback is used in conjunction with training using model.fit()
to save a model or weights (in a checkpoint file) at some interval,
#so the model or weights can be loaded later to continue the training from the
state saved.
checkpoint =
ModelCheckpoint("model.h5",monitor="val_accuracy",save_best_only=True,mode="auto
",verbose=1)

#Stop training when a monitored metric has stopped improving.
earlystop =
EarlyStopping(monitor="val_accuracy",patience=5,mode="auto",verbose=1)
```

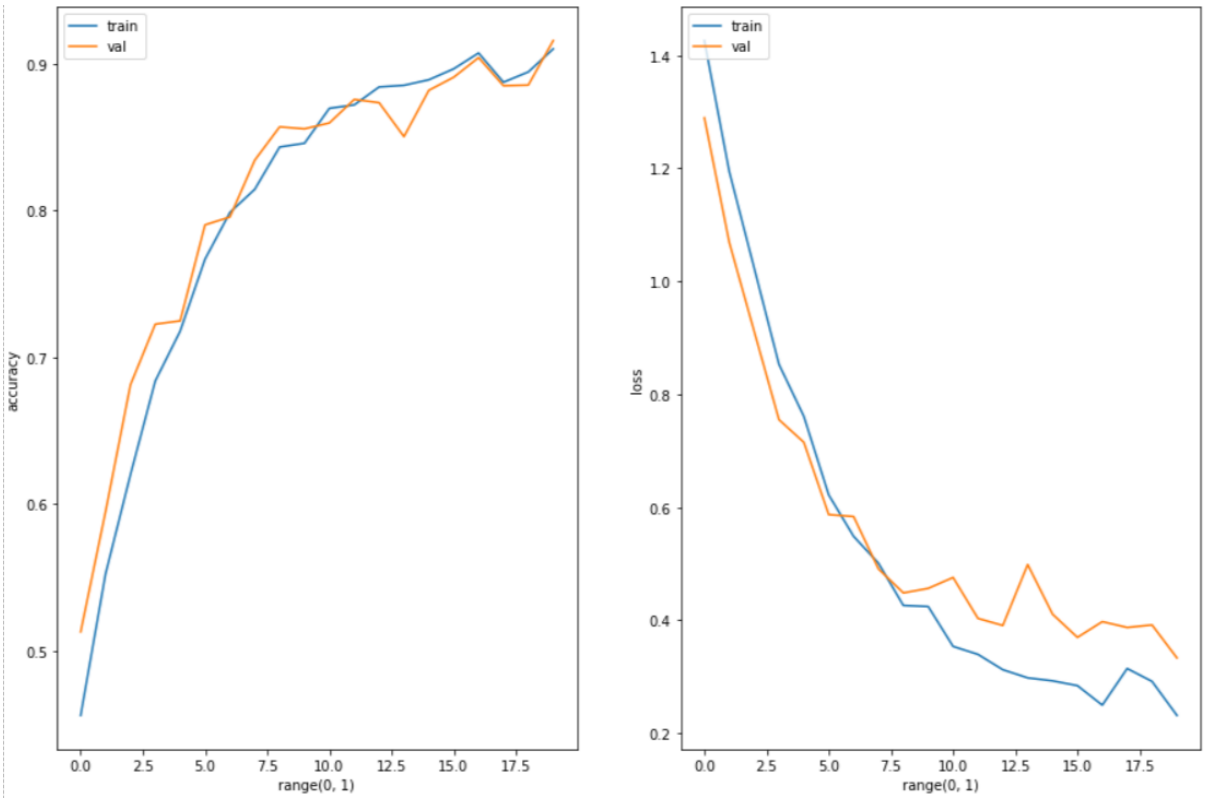
Train the Model:

```
# Train the model
epochs = 20
history = model.fit(train_ds, validation_data=val_ds,
epochs=epochs,callbacks=[checkpoint,earlystop])
```

```
Epoch 1/20
281/281 [==============================] - ETA: 0s - loss: 1.4257 - accuracy: 0.4557
Epoch 1: val_accuracy improved from -inf to 0.51268, saving model to model.h5
281/281 [==============================] - 221s 788ms/step - loss: 1.4257 - accuracy: 0.4557 - val_loss: 1.2891 - val_accuracy:
0.5127
Epoch 2/20
281/281 [==============================] - ETA: 0s - loss: 1.1940 - accuracy: 0.5522
Epoch 2: val_accuracy improved from 0.51268 to 0.59457, saving model to model.h5
281/281 [==============================] - 210s 747ms/step - loss: 1.1940 - accuracy: 0.5522 - val_loss: 1.0687 - val_accuracy:
0.5946
Epoch 3/20
281/281 [==============================] - ETA: 0s - loss: 1.0237 - accuracy: 0.6197
Epoch 3: val_accuracy improved from 0.59457 to 0.68091, saving model to model.h5
281/281 [==============================] - 205s 729ms/step - loss: 1.0237 - accuracy: 0.6197 - val_loss: 0.9116 - val_accuracy:
0.6809
Epoch 4/20
281/281 [==============================] - ETA: 0s - loss: 0.8527 - accuracy: 0.6836
Epoch 4: val_accuracy improved from 0.68091 to 0.72230, saving model to model.h5
281/281 [==============================] - 201s 716ms/step - loss: 0.8527 - accuracy: 0.6836 - val_loss: 0.7549 - val_accuracy:
0.7223
Epoch 5/20
281/281 [==============================] - ETA: 0s - loss: 0.7607 - accuracy: 0.7175
Epoch 5: val_accuracy improved from 0.72230 to 0.72452, saving model to model.h5
281/281 [==============================] - 206s 732ms/step - loss: 0.7607 - accuracy: 0.7175 - val_loss: 0.7148 - val_accuracy:
0.7245
Epoch 6/20
281/281 [==============================] - ETA: 0s - loss: 0.6218 - accuracy: 0.7666
Epoch 6: val_accuracy improved from 0.72452 to 0.78994, saving model to model.h5
281/281 [==============================] - 208s 739ms/step - loss: 0.6218 - accuracy: 0.7666 - val_loss: 0.5869 - val_accuracy:
0.7899
```

Plot the Training Curves:

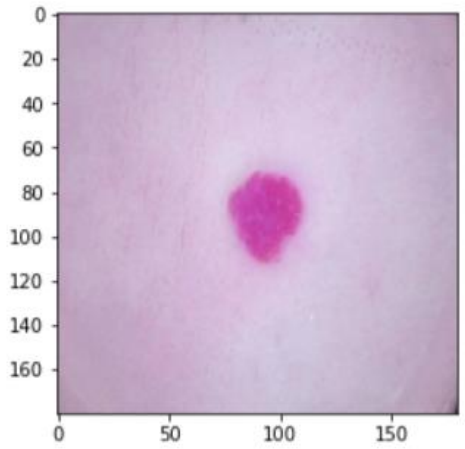
```
# Plot the training curves

epochs_range = range(earlystop.stopped_epoch+1)

plt.figure(figsize=(15, 10))
plt.subplot(1, 2, 1)

#Plot Model Accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel(epochs_range)
plt.legend(['train', 'val'], loc='upper left')

#Plot Model Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel(epochs_range)
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



Model Prediction(Result):

```
from glob import glob
Test_image_path = os.path.join(data_dir_test, class_names[1], '*')
Test_image = glob("C:\\Users\\dassa\\Downloads\\Skin cancer ISIC The
International Skin Imaging Collaboration\\Test\\ISIC_0024402.jpg")
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[1] +'\n'+ "Predictive Class "+pred_class )
```

```
1/1 [==============================] - 0s 31ms/step
Actual Class basal cell carcinoma
Predictive Class vascular lesion
```



INFERENCE

We trained this model to detect the skin cancer using Convolutional Neural Network model. From this project we able to learn about the Artificial Neural Network architecture and its working. It makes us understand about the CNN model’s layers and its working technique. Train the model using a large dataset makes the model more efficient and effective, which yields a precise result . We get precise result after successfully train the model for 20 epochs. So, Finally we are able to build a model which can predict skin cancer at an accuracy of 90.98% And a Validation accuracy of 91.54%.

REFERENCES

1. Edureka – Cancer Detection using deep learning: https://youtu.be/7MceDfpnP8k
2. A comprehensive guide to convolutional Neural Networks – https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53