*Skin Cancer Classification with Mobile Net*

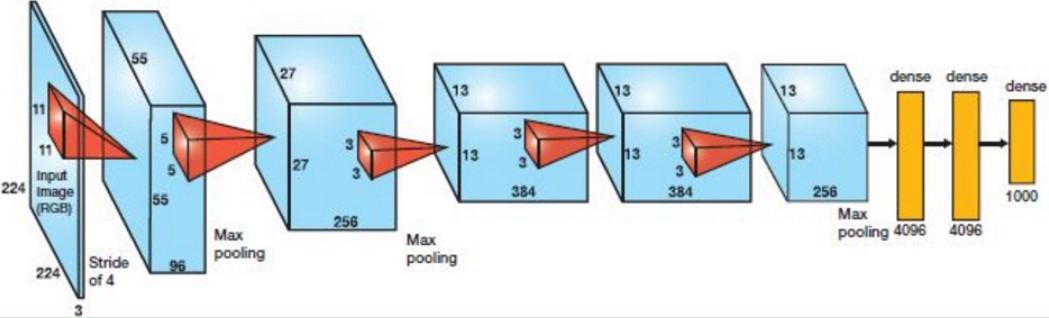
*Paper name:*Skin Lesion Analyzer : An Efficient Seven-Way Multi-class Skin Cancer Classification Using Mobile Net 2020

*Authors:*Saket S. Chaturvedi , Kaj0ol Gupta, and Prakash S. Prasad Abstract

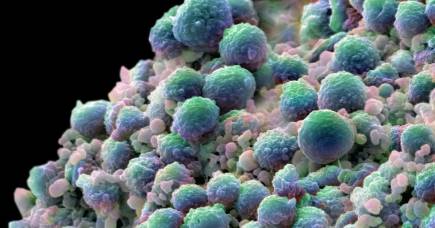
Data Set: <https://www.kaggle.com/datasets/nodoubttome/skin-cancer9-classesisic>

Publisher: [International Conference on Advanced Machine Learning Technologies and Applications](https://link.springer.com/conference/amlta%20amlta) , book: [**Advanced Machine Learning Technologies and Applications**](https://link.springer.com/book/10.1007/978-981-15-3383-9)

Algorithm: Mobile Net v2 Based on Transfer Learning



[This Photo](https://www.pianshen.com/article/3584423646/) by Unknown Author is licensed under [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)



[This Photo](https://pursuit.unimelb.edu.au/articles/gearing-up-against-skin-cancer) by Unknown Author is licensed under [CC BY-ND](https://creativecommons.org/licenses/by-nd/3.0/)

Overview: Provide a brief overview of the model, including its purpose, the problem it is designed to solve, and any key features or benefits

Skin cancer classification with Mobile Net is a deep learning model that is designed to classify skin lesions into different types of skin cancer, including

('actinic keratosis', 'basal cell carcinoma', 'dermatofibroma', 'melanoma', 'nevus', 'pigmented benign keratosis', 'seborrheic keratosis', 'squamous cell carcinoma', 'vascular lesion')

. The purpose of the model is to assist dermatologists and other medical professionals in diagnosing skin cancer more accurately and efficiently, by providing an automated classification tool that can analyze images of skin lesions and provide a diagnosis.

The problem that the model is designed to solve is the difficulty of accurately diagnosing skin cancer based on visual inspection alone. Dermatologists and other medical professionals typically rely on visual inspection of skin lesions to diagnose skin cancer, but this approach can be subjective and prone to errors. By using deep learning to analyze images of skin lesions, the Mobile Net model can provide a more objective and accurate diagnosis, which can help improve patient outcomes and reduce unnecessary biopsies and surgeries.

One key feature of the Mobile Net model is its use of transfer learning, which allows the model to leverage pre-trained weights from other deep learning models to improve its performance. Mobile Net is a lightweight deep learning architecture that is optimized for mobile devices, making it well-suited for use in mobile applications and other resource-constrained environments. The model can be trained on large datasets of skin lesion images to improve its accuracy and generalizability, and can be fine-tuned for specific use cases and applications.

Overall, skin cancer classification with Mobile Net is a promising approach to improving the accuracy and efficiency of skin cancer diagnosis, and has the potential to improve patient outcomes and reduce healthcare costs

*and we will implement the details of this data set with our model to give the highest needs for our* dermatologists *…*

**Intro for Skin Cancer :**

**Abstract** Skin cancer is an emerging global health problem with 123,000 melanoma and 3,000,000 non-melanoma cases worldwide each year. The recent studies have reported excessive exposure to ultraviolet rays as a major factor in developing skin cancer. The most effective solution to control the death rate for skin cancer is a timely diagnosis of skin lesions as the five-year survival rate for melanoma patients is 99% when diagnosed and screened at the early stage. Considering an inability of dermatologists for accurate diagnosis of skin cancer, there is a need to develop an automated efficient system for the diagnosis of skin cancer. This study explores an efficient automated method for skin cancer classification with better evaluation metrics as compared to previous studies or expert dermatologists. We utilized a Mobile Net model pretrained on approximately 1,280,000 images from 2014 ImageNet Challenge and finetuned on 10,015 dermoscopy images of skin-cancer9-classesisic/Skin cancer ISIC The International Skin Imaging Collaboration dataset employing transfer learning. The model used in this study achieved an overall accuracy for the Testing it near about **51.01%** for 9 classes in the dataset, whereas top2 and top3 accuracies of **76.27%** and **88.14%**, respectively. Also, the weighted average of precision, weighted

average of recall, and weighted average of f1-score were found to be **97.97%**, **99.77%,** and **83.56%**, respectively for the validation data. This method has the potential to assist dermatology specialists prevalence of harmful ultraviolet rays in the earth’s environment. The researchers had discovered a further 10% depletion of the ozone layer will intensify the problem of skin cancer with an additional 300,000 non-melanoma and 4,500 melanoma cases each year . Currently,

every year, 123,000 melanomas and 3,000,000 non-melanoma

cases are recorded worldwide. The recent study on the prevention of skin cancer

reports 90% of non-melanoma and 86% of melanoma cases induced by excessive

exposure of ultraviolet rays . The UV radiation detriments the DNA present at

the inner layers of skin, triggering the uncontrolled growth of skin cells, which may even emerge as a skin cancer. The most straightforward and effective solution to control the mortality rate for skin cancer is the timely diagnosis of skin cancer as the survival rate for melanoma patients in a five-year timespan is 99% when diagnosed and screened at the early stage . Moreover, the most mundane skin cancer types BCC and SCC are highly treatable when early diagnosed and treated adequately . Dermatologist primarily utilizes visual inspection to diagnose skin cancer, which is a challenging task considering the visual similarity among skin cancers. However, dermoscopy has been popular for the diagnosis of skin cancer recently considering the ability of dermoscopy to accurately visualize the skin lesions not discernible with the naked eye. Reports on the diagnostic accuracy of clinical dermatologists have claimed 80% diagnostic accuracy for a dermatologist with experience greater than ten years, whereas the dermatologists with experience of 3–5 years were able to achieve diagnostic accuracy of only 62%, the accuracy further dropped for less-experienced dermatologists The studies on Dermoscopy imply a need to develop an automated efficient

and robust system for the diagnosis of skin cancer since the fledgling dermatologists may deteriorate the diagnostic accuracy of skin lesions

Although the method is complicated, deep learning algorithms have shown exceptional performance in visual tasks and even outperformed humans in gaming, , Atari and object recognition , which has lead to conduct the research on automated screening of skin cancers . Several studies have been done to compare the dermatologist level, and Deep learning-based automated classification of

skin cancer Esteva et al. reported a benchmark study comparing the performance

of dermatologists and a CNN model over 129,450 clinical images, showing

the CNN model performs at par or better than dermatologists . In recent years,

the trend has shifted to deep neural networks (DNNs) , which were proposed

to overcome the drawbacks of previous models . Although DNNs require

huge data for the training, they have an appealing impact on medical image classification The current literature mostly employs transfer learning to solve large

dataset problem. Transfer learning is a method where a model trained over another

knowledge from ImageNet. The main difference between the DNN architecture

and implementation framework—Caffe is the most common framework,

and Res Net , Alex Net , VGG-16 are most common architectures.

Previous work in dermoscopic-automated skin cancer classification has lacked

generality capability ,and have not achieved pleasing results for multiclass

skin cancer classification. This study explores an efficient automated

method for the classification of dermoscopy skin cancer images.We utilized a

Mobile Net convolutional neural network pretrained on approximately 1,280,000

images from 2014 ImageNet Challenge and finetuned on skin-cancer9-classesisic/Skin cancer ISIC The International Skin Imaging Collaboration dataset which contain 10,015 dermoscopy images employing transfer learning

The Mobile Net model classified skin lesion image with performance better or comparable to expert dermatologists for seven classes. We also conducted data analysis on the dermoscopy images of skin cancer from skin-cancer9 classesisic/Skin cancer ISIC The International Skin Imaging Collaboration dataset to uncover the relation of skin cancer with several parameters to strengthen the understanding of skin cancer.

Into About Transfer Learning :

Architecture: Describe the architecture of the model, including the layers, activation functions, and any other key components. Provide diagrams or visualizations to help users understand the model. First will use Transfer Learning approach for our Problem with learning And we will take a brief about it:

When you're building a computer vision application, you can build your convnets as we learned in chapter three and start the training from scratch. And that is an acceptable approach. Another much faster approach is to download a neural network that someone else has already built and trained on a large dataset in a certain domain and use this pretrained network as a starting point to train the network on your new task. This approach is called **transfer** **learning**. Transfer learning is one of the most important techniques of deep learning. When building a vision system to solve a specific problem, you usually need to collect and label a huge amount of data to train your network. But what if we could use an existing neural network, that someone else has tuned and trained, and use it as a starting point for our new task? Transfer learning allows us to do just that. We can download an open-source model that someone else has already trained and tuned for weeks and use their optimized parameters (weights) as a starting point to train our model just a little bit more on a smaller dataset that we have for a given task. This way we can train our network a lot faster and achieve very high results. Deep learning researchers and practitioners have posted a lot of research papers and opensource projects of their trained algorithms that they have worked on for weeks and months and trained on many GPUs to get state-of-the-art results on many problems. The fact that someone else has done this work and gone through the painful high-performance research process, means that you can often download open source architecture and weights that took someone else many weeks or months to build and tune and use that as a very good start for your own neural network. This is **transfer learning.** It is referring to the knowledge transfer from pretrained network in one domain to your own problem in a different domain

**What are the problems that transfer learning is solving**:

**Data problem:** it requires a lot of data to be able to get decent results which is not

very feasible in most cases. It is relatively rare to have a dataset of sufficient size to

solve your problem. It is also very expensive to acquire and label data which is mostly

a manual process that has to be done by humans capturing images and labeling them

one-by-one which makes it a non-trivial, very expensive task.

**Computation problem:** even if you are able to acquire hundreds of thousands of

images for your problem, it is computationally very expensive to train a deep neural

network on millions of images. The training process of a deep neural network from

scratch is very expensive because it usually requires weeks of training on multiple

GPUs. Also Keep in mind that the neural network training process is an iterative

process. So, even if you happen to have the computing power that is needed to train

complex neural networks, having to spend a few weeks experimenting different

hyperparameters in each training iteration will make the project very expensive until

you finally reach satisfactory results

*How we can use Transfer Learning :*

1- Decide how much to retrain: Depending on the size and complexity of your dataset, you may need to retrain only the final layer of the pre-trained model, or you may need to retrain more layers.

2- Prepare your dataset: You will need to prepare your dataset by splitting it into training, validation, and testing sets, and preprocessing the images or data as necessary. You may also need to augment the dataset with techniques like rotation, scaling, and flipping to increase the diversity of the data.

3- Retrain the model: Once you have prepared your dataset, you can retrain the pre-trained model using your dataset. The exact approach will depend on the framework you are using, but typically involves freezing the weights of the pre-trained layers and training the final layer or layers on your dataset.

4- Fine-tune the model: After retraining the model on your dataset, you can fine-tune the model by unfreezing some of the pre-trained layers and retraining them on your dataset. This can help improve the performance of the model by allowing it to adapt more closely to your specific task.

5- Evaluate the model: Finally, you can evaluate the performance of the model on a separate test set to determine its accuracy and other performance metrics.

6- Transfer learning can be a powerful technique for reducing the amount of time and resources required to train a deep learning model, especially when working with limited data or computational resources. By using a pre-trained model as a starting point, you can leverage the knowledge and expertise that has been built into the model, while customizing it to your specific task.

*Technique Used For Transfer Learning :*

1. Feature extraction: In this approach, the pre-trained model is used as a fixed feature extractor, and the output features of the pre-trained model are used as input to a new model. The new model is trained only on the new dataset, and the pre-trained model is frozen and not updated during training.
2. Fine-tuning: In this approach, the pre-trained model is used as a starting point, and the weights of some of the layers are updated during training on the new dataset. This approach requires more labeled data and can be prone to overfitting.
3. One-shot learning: In this approach, the pre-trained model is used to learn the similarity between different classes in a new dataset, and then a new model is trained on a small amount of labeled data per class.
4. Domain adaptation: In this approach, the pre-trained model is adapted to a new domain by fine-tuning on a small amount of labeled data from the new domain.

Model

1-Intro About Datasets:

This set consists of 2357 images of malignant and benign oncological diseases, which were formed from The International Skin Imaging Collaboration (ISIC). All images were sorted according to the classification taken with ISIC, and all subsets were divided into the same number of images, with the exception of melanomas and moles, whose images are slightly dominant.

The data set contains the following diseases:

1. actinic keratosis
2. basal cell carcinoma
3. dermatofibroma
4. melanoma
5. nevus
6. pigmented benign keratosis9
7. seborrheic keratosis
8. squamous cell carcinoma
9. vascular lesion

2- Importing Essential Libraries

the model based on tensorflow and it is related packages

Code:

import os

import pathlib

import shutil

import cv2

import gc

import keras

import numpy as np

import pandas as pd

from keras.applications.mobilenet import MobileNet

from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

from keras.layers import (BatchNormalization, Dense, Dropout, Flatten)

from keras.metrics import categorical\_accuracy, top\_k\_categorical\_accuracy

from keras.models import Sequential

from keras.preprocessing.image import ImageDataGenerator

from keras.optimizers import Adam

from matplotlib import pyplot as plt

from sklearn.metrics import confusion\_matrix, f1\_score

from sklearn.model\_selection import train\_test\_split

import random

from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D

\*\*\*\*\*

2- Importing Data

from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():

  print('User uploaded file "{name}" with length {length} bytes'.format(

      name=fn, length=len(uploaded[fn])))

# Then move kaggle.json into the folder where the API expects to find it.

!mkdir -p ~/.kaggle/ && mv kaggle.json ~/.kaggle/ && chmod 600 ~/.kaggle/kaggle.json

\*\*\*\*

!kaggle datasets download -d nodoubttome/skin-cancer9-classesisic

Output:

Downloading skin-cancer9-classesisic.zip to /content

100% 786M/786M [00:38<00:00, 22.1MB/s]

100% 786M/786M [00:38<00:00, 21.2MB/s]

Extract the content of the folder to be the data set

from zipfile import ZipFile

file\_name = "/content/skin-cancer9-classesisic.zip"

with ZipFile(file\_name,'r') as zip:

  zip.extractall()

  print('Done')

* Done

!pip install split-folders

Output:

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>

Collecting split-folders

Downloading split\_folders-0.5.1-py3-none-any.whl (8.4 kB)

Installing collected packages: split-folders

Successfully installed split-folders-0.5.1

Splitting the folders into:

import splitfolders

splitfolders.ratio("/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train", # The location of dataset

                   output="{output\_directory}", # The output location

                   seed=42, # The number of seed

                   ratio=(.7,.2,.1), # The ratio of splited dataset

                   group\_prefix=None, # If your dataset contains more than one file like ".jpg", ".pdf", etc

                   move=False # If you choose to move, turn this into True

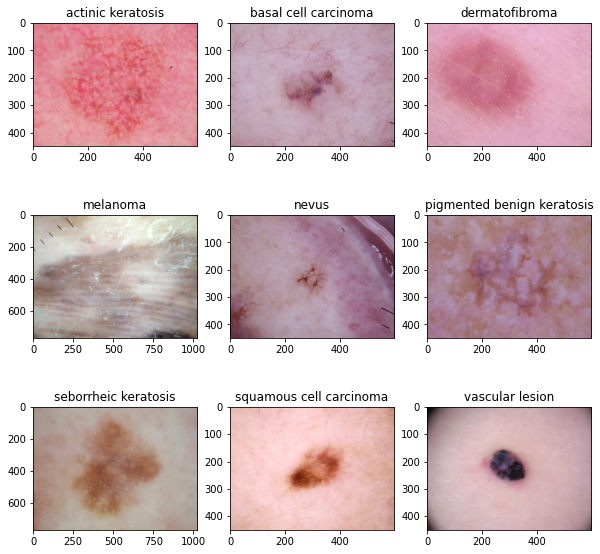
                   )

* copying files: 2239 files

and this the final code in the step of importing the data into folders

4- ***Data Preprocessing***

|  |  |
| --- | --- |
| Train | Test |
| Actinic keratosis 114 files | Actinic keratosis 16 files |
| Basal cell carcinoma 376 files | Basal cell carcinoma 16 files |
| Melanoma 438 files | Melanoma 16 files |
| Nevus 357 files | Nevus files |
| Pigmented Benign Keratosis 462 files | Pigmented Benign Keratosis 16 files |
| Seborrheic Keratosis 77 files | Seborrheic Keratosis 16 files |
| Squamous cell carcinoma 181 files | Squamous cell carcinoma files |
| Dermatofibroma 95 files | Dermatofibroma files |
| Vascular lesion 139 files | Vascular lesion 3 files |
|  |  |



In this Step we will talk about more details……

# Declaration the Train , Test and Validation

data\_dir\_train = pathlib.Path("/content/Skin cancer ISIC The International Skin Imaging Collaboration/Train")

data\_dir\_test = pathlib.Path("/content/Skin cancer ISIC The International Skin Imaging Collaboration/Test")

data\_dir\_val = pathlib.Path("/content/{output\_directory}/val")

image\_count\_train = len(list(data\_dir\_train.glob('\*/\*.jpg')))

print(image\_count\_train)

image\_count\_test = len(list(data\_dir\_test.glob('\*/\*.jpg')))

print(image\_count\_test)

image\_count\_val = len(list(data\_dir\_val.glob('\*/\*.jpg')))

print(image\_count\_val)

output:

*2239 118 444*

# define the Class Names (Labels)

class\_names = ['actinic keratosis', 'basal cell carcinoma', 'dermatofibroma', 'melanoma', 'nevus', 'pigmented benign keratosis', 'seborrheic keratosis', 'squamous cell carcinoma', 'vascular lesion']

# Calculates the number of images per Labels for our data set

for label in class\_names:

    print(label + " train: " + str(len(os.listdir(os.path.join(data\_dir\_train, label)))))

print("\n")

for label in class\_names:

    print(label + " val: " + str(len(os.listdir(os.path.join(data\_dir\_val, label)))))

print("\n")

for label in class\_names:

    print(label + " test: " + str(len(os.listdir(os.path.join(data\_dir\_test, label)))))

output:

actinic keratosis train: 114

basal cell carcinoma train: 376

dermatofibroma train: 95

melanoma train: 438

nevus train: 357

pigmented benign keratosis train: 462

seborrheic keratosis train: 77

squamous cell carcinoma train: 181

vascular lesion train: 139

actinic keratosis val: 22

basal cell carcinoma val: 75

dermatofibroma val: 19

melanoma val: 87

nevus val: 71

pigmented benign keratosis val: 92

seborrheic keratosis val: 15

squamous cell carcinoma val: 36

vascular lesion val: 27

actinic keratosis test: 16

basal cell carcinoma test: 16

dermatofibroma test: 16

melanoma test: 16

nevus test: 16

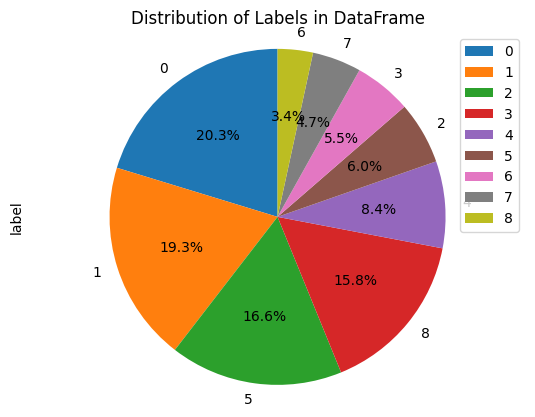
pigmented benign keratosis test: 16

seborrheic keratosis test: 3

squamous cell carcinoma test: 16

vascular lesion test: 3

end of our code for the part of Preprocessing

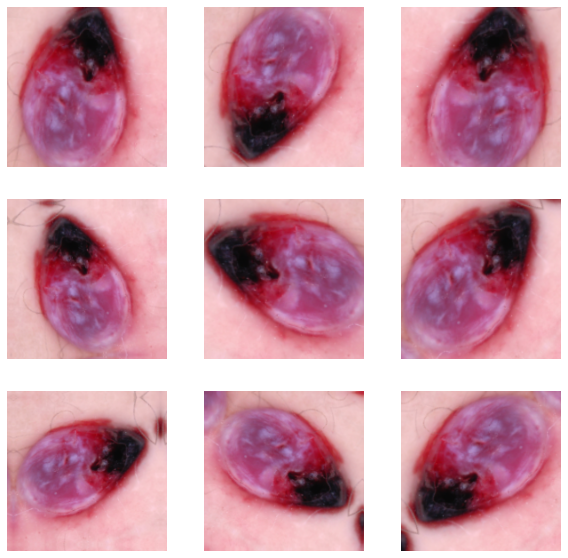
and this the

This is just A pie chart for information or details about the data set in a analysis way..

3- ***Data Augmentation***

***in this part we will talk about the most important part of avoid the overfit in the mode by Augmentation to our Data set we will balance the labels at our model to be more suitable for the best results that we will generate***

***we used*** Image Data Generator

******

data\_gen\_param = {

    "rotation\_range": 180,

    "width\_shift\_range": 0.1,

    "height\_shift\_range": 0.1,

    "zoom\_range": 0.1,

    "horizontal\_flip": True,

    "vertical\_flip": True

}

data\_generator = ImageDataGenerator(\*\*data\_gen\_param)

num\_images\_each\_label = 6000

base\_dir =pathlib.Path("/content/Skin cancer ISIC The International Skin Imaging Collaboration")

aug\_dir2 = os.path.join( base\_dir, "aug\_dir2")

os.mkdir(aug\_dir2)

for label in class\_names:

    img\_dir = os.path.join(aug\_dir2, "aug\_img")

    os.mkdir(img\_dir)

    src\_dir\_label = os.path.join(data\_dir\_train, label)

    for image\_name in os.listdir(src\_dir\_label):

        shutil.copy(os.path.join(src\_dir\_label, image\_name), os.path.join(img\_dir, image\_name))

    batch\_size = 32

    data\_flow\_param = {

  "directory": aug\_dir2,

"color\_mode": "rgb",

        "batch\_size": batch\_size,

        "shuffle": True,

        "save\_to\_dir": os.path.join(data\_dir\_train, label),

        "save\_format": "jpg"

    }

    aug\_data\_gen = data\_generator.flow\_from\_directory(\*\*data\_flow\_param)

    num\_img\_aug = num\_images\_each\_label - len(os.listdir(os.path.join(data\_dir\_train, label)))

    num\_batch = int(num\_img\_aug / batch\_size)

    for i in range(0, num\_batch):

        next(aug\_data\_gen)

Ouput:

Found 114 images belonging to 1 classes.

Found 376 images belonging to 1 classes.

Found 95 images belonging to 1 classes.

Found 438 images belonging to 1 classes.

Found 357 images belonging to 1 classes.

Found 462 images belonging to 1 classes.

Found 77 images belonging to 1 classes.

Found 181 images belonging to 1 classes.

Found 139 images belonging to 1 classes.

More Details about the code :

The data\_gen\_param dictionary defines the parameters for data augmentation, including rotation range, width shift range, height shift range, zoom range, horizontal flip, and vertical flip.

The ImageDataGenerator object is created with these parameters using the \*\* operator, which unpacks the dictionary as keyword arguments.

The aug\_dir2 directory is created to store the augmented images.

For each class in the class\_names list, a new directory is created in the aug\_dir2 directory to store the augmented images.

The original images for the current class are copied from the data\_dir\_train directory to the new directory created in step 4.

The data\_flow\_param dictionary is created with the parameters for the flow\_from\_directory method of the ImageDataGenerator object. This dictionary includes the path to the directory containing the images, the batch size, and other parameters.

The flow\_from\_directory method is called with the data\_flow\_param dictionary to create a generator object that yields batches of augmented images.

The number of augmented images to generate is calculated by subtracting the number of original images from num\_images\_each\_label, which is a variable that specifies the desired number of images for each class.

The number of batches needed to generate the desired number of images is calculated by dividing the number of augmented images by the batch size.

A loop is used to generate the augmented images in batches. The next method is called on the generator object for each batch, which generates and saves the augmented images to the directory specified in data\_flow\_param.

By generating augmented images for each class in the training dataset, the model is exposed to a wider variety of training examples and is less likely to overfit to the existing data. The resulting model is expected to generalize better to new examples.

IMAGE\_SHAPE = (224, 224, 3)

data\_gen\_param = {

    "samplewise\_center": True,

    "samplewise\_std\_normalization": True,

    "rotation\_range": 180,

    "width\_shift\_range": 0.1,

    "height\_shift\_range": 0.1,

    "zoom\_range": 0.1,

    "horizontal\_flip": True,

    "vertical\_flip": True,

    "rescale": 1.0 / 255

}

data\_generator = ImageDataGenerator(\*\*data\_gen\_param)

train\_flow\_param = {

    "directory": data\_dir\_train,

    "batch\_size": 64,

    "target\_size": IMAGE\_SHAPE[:2],

    "shuffle": True,

}

train\_flow = data\_generator.flow\_from\_directory(\*\*train\_flow\_param)

val\_flow\_param = {

    "directory": data\_dir\_val,

    "batch\_size": 1,

    "target\_size": IMAGE\_SHAPE[:2],

    "shuffle": False

}

val\_flow = data\_generator.flow\_from\_directory(\*\*val\_flow\_param)

test\_flow\_param = {

    "directory": data\_dir\_test,

    "batch\_size": 1,

    "target\_size": IMAGE\_SHAPE[:2],

    "shuffle": False

}

test\_flow = data\_generator.flow\_from\_directory(\*\*test\_flow\_param)

output:

Found 50111 images belonging to 9 classes.

Found 444 images belonging to 9 classes.

Found 118 images belonging to 9 classes.

More Detail about the code..

This code initializes ImageDataGenerator objects for data augmentation and creates generator objects for the training, validation, and test datasets using these ImageDataGenerator objects. Here's how the code works:

IMAGE\_SHAPE is a tuple that defines the shape of the images, with dimensions 224x224 and 3 color channels (RGB). data\_gen\_param is a dictionary that defines the parameters for data augmentation, including centering the data to have zero mean, scaling the data to have unit variance, rotating the image up to 180 degrees, shifting the width and height of the image up to 10%, zooming the image up to 10%, flipping the image horizontally and vertically, and rescaling the pixel values to be between 0 and 1.The ImageDataGenerator object is created with these parameters using the \*\* operator, which unpacks the dictionary as keyword arguments.The train\_flow\_param dictionary is created with the parameters for the flow\_from\_directory method of the ImageDataGenerator object. This dictionary includes the path to the directory containing the training images, the batch size, the target image size, and other parameters.The flow\_from\_directory method is called with the train\_flow\_param dictionary to create a generator object that yields batches of augmented training images. The val\_flow\_param dictionary is created in a similar way to create a generator object for the validation dataset. The test\_flow\_param dictionary is created in a similar way to create a generator object for the test dataset.

By generating generator objects for each dataset using the ImageDataGenerator object with data augmentation parameters, the model is trained on a wider variety of training examples and is less likely to overfit to the existing data. The resulting model is expected to generalize better to new examples

4- Building Architecture and Training Algorithm

This heart of our model that we want to build our model based on Transfer Learning

We used Mobile net in our mode with Base Model that contain about 3,228,864

and Trainable params: 3,206,976 , Non-trainable params: 21,888

and we have a some of details about the architecture The MobileNet model is ideal for mobile and embedded vision applications as they

have lightweight DNN architecture . We used Mobile Net convolutional neural

network pretrained on 1,280,000 images containing 1,000 object classes from

the 2014 ImageNet Challenge. The 25 layered Mobile Net architecture was

constructed for the current study, which employs four Conv2D layers, seven Batch-

Normalization layers, seven ReLU layers, three ZeroPadding2D layers, and single

DepthwiseConv2D, GlobalAveragePooling, Dropout, and Dense layers as shown

in Fig The training of the model was done on a training set of 38,569 images

using transfer learning with batch size and epochs as 10 and 50, respectively.

The categorical crossentropy loss function, Adam optimizer, and metric function

accuracy, top2 accuracy, and top3 accuracy were used to evaluate MobileNet model

performance.

And this are the details of parameter for the main Architecture for our model:

Input Operator t c n s

2242 \_ 3 conv2d - 32 1 2

1122 \_ 32 bottleneck 1 16 1 1

1122 \_ 16 bottleneck 6 24 2 2

562 \_ 24 bottleneck 6 32 3 2

282 \_ 32 bottleneck 6 64 4 2

142 \_ 64 bottleneck 6 96 3 1

142 \_ 96 bottleneck 6 160 3 2

72 \_ 160 bottleneck 6 320 1 1

72 \_ 320 conv2d 1x1 - 1280 1 1

72 \_ 1280 avgpool 7x7 - - 1 -

1 \_ 1 \_ 1280 conv2d 1x1 - k -

Table 2: MobileNetV2 : Each line describes a sequence

Block Diagram for the model:

Model: "mobilenet\_1.00\_224"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_1 (InputLayer) [(None, 224, 224, 3)] 0

conv1 (Conv2D) (None, 112, 112, 32) 864

conv1\_bn (BatchNormalizatio (None, 112, 112, 32) 128

n)

conv1\_relu (ReLU) (None, 112, 112, 32) 0

conv\_dw\_1 (DepthwiseConv2D) (None, 112, 112, 32) 288

conv\_dw\_1\_bn (BatchNormaliz (None, 112, 112, 32) 128

ation)

conv\_dw\_1\_relu (ReLU) (None, 112, 112, 32) 0

conv\_pw\_1 (Conv2D) (None, 112, 112, 64) 2048

conv\_pw\_1\_bn (BatchNormaliz (None, 112, 112, 64) 256

ation)

conv\_pw\_1\_relu (ReLU) (None, 112, 112, 64) 0

conv\_pad\_2 (ZeroPadding2D) (None, 113, 113, 64) 0

conv\_dw\_2 (DepthwiseConv2D) (None, 56, 56, 64) 576

conv\_dw\_2\_bn (BatchNormaliz (None, 56, 56, 64) 256

ation)

conv\_dw\_2\_relu (ReLU) (None, 56, 56, 64) 0

conv\_pw\_2 (Conv2D) (None, 56, 56, 128) 8192

conv\_pw\_2\_bn (BatchNormaliz (None, 56, 56, 128) 512

ation)

conv\_pw\_2\_relu (ReLU) (None, 56, 56, 128) 0

conv\_dw\_3 (DepthwiseConv2D) (None, 56, 56, 128) 1152

conv\_dw\_3\_bn (BatchNormaliz (None, 56, 56, 128) 512

ation)

conv\_dw\_3\_relu (ReLU) (None, 56, 56, 128) 0

conv\_pw\_3 (Conv2D) (None, 56, 56, 128) 16384

conv\_pw\_3\_bn (BatchNormaliz (None, 56, 56, 128) 512

ation)

conv\_pw\_3\_relu (ReLU) (None, 56, 56, 128) 0

conv\_pad\_4 (ZeroPadding2D) (None, 57, 57, 128) 0

conv\_dw\_4 (DepthwiseConv2D) (None, 28, 28, 128) 1152

conv\_dw\_4\_bn (BatchNormaliz (None, 28, 28, 128) 512

ation)

conv\_dw\_4\_relu (ReLU) (None, 28, 28, 128) 0

conv\_pw\_4 (Conv2D) (None, 28, 28, 256) 32768

conv\_pw\_4\_bn (BatchNormaliz (None, 28, 28, 256) 1024

ation)

conv\_pw\_4\_relu (ReLU) (None, 28, 28, 256) 0

conv\_dw\_5 (DepthwiseConv2D) (None, 28, 28, 256) 2304

conv\_dw\_5\_bn (BatchNormaliz (None, 28, 28, 256) 1024

ation)

conv\_dw\_5\_relu (ReLU) (None, 28, 28, 256) 0

conv\_pw\_5 (Conv2D) (None, 28, 28, 256) 65536

conv\_pw\_5\_bn (BatchNormaliz (None, 28, 28, 256) 1024

ation)

conv\_pw\_5\_relu (ReLU) (None, 28, 28, 256) 0

conv\_pad\_6 (ZeroPadding2D) (None, 29, 29, 256) 0

conv\_dw\_6 (DepthwiseConv2D) (None, 14, 14, 256) 2304

conv\_dw\_6\_bn (BatchNormaliz (None, 14, 14, 256) 1024

ation)

conv\_dw\_6\_relu (ReLU) (None, 14, 14, 256) 0

conv\_pw\_6 (Conv2D) (None, 14, 14, 512) 131072

conv\_pw\_6\_bn (BatchNormaliz (None, 14, 14, 512) 2048

ation)

conv\_pw\_6\_relu (ReLU) (None, 14, 14, 512) 0

conv\_dw\_7 (DepthwiseConv2D) (None, 14, 14, 512) 4608

conv\_dw\_7\_bn (BatchNormaliz (None, 14, 14, 512) 2048

ation)

conv\_dw\_7\_relu (ReLU) (None, 14, 14, 512) 0

conv\_pw\_7 (Conv2D) (None, 14, 14, 512) 262144

conv\_pw\_7\_bn (BatchNormaliz (None, 14, 14, 512) 2048

ation)

conv\_pw\_7\_relu (ReLU) (None, 14, 14, 512) 0

conv\_dw\_8 (DepthwiseConv2D) (None, 14, 14, 512) 4608

conv\_dw\_8\_bn (BatchNormaliz (None, 14, 14, 512) 2048

ation)

conv\_dw\_8\_relu (ReLU) (None, 14, 14, 512) 0

conv\_pw\_8 (Conv2D) (None, 14, 14, 512) 262144

conv\_pw\_8\_bn (BatchNormaliz (None, 14, 14, 512) 2048

ation)

conv\_pw\_8\_relu (ReLU) (None, 14, 14, 512) 0

conv\_dw\_9 (DepthwiseConv2D) (None, 14, 14, 512) 4608

conv\_dw\_9\_bn (BatchNormaliz (None, 14, 14, 512) 2048

ation)

conv\_dw\_9\_relu (ReLU) (None, 14, 14, 512) 0

conv\_pw\_9 (Conv2D) (None, 14, 14, 512) 262144

conv\_pw\_9\_bn (BatchNormaliz (None, 14, 14, 512) 2048

ation)

conv\_pw\_9\_relu (ReLU) (None, 14, 14, 512) 0

conv\_dw\_10 (DepthwiseConv2D (None, 14, 14, 512) 4608

)

conv\_dw\_10\_bn (BatchNormali (None, 14, 14, 512) 2048

zation)

conv\_dw\_10\_relu (ReLU) (None, 14, 14, 512) 0

conv\_pw\_10 (Conv2D) (None, 14, 14, 512) 262144

conv\_pw\_10\_bn (BatchNormali (None, 14, 14, 512) 2048

zation)

conv\_pw\_10\_relu (ReLU) (None, 14, 14, 512) 0

conv\_dw\_11 (DepthwiseConv2D (None, 14, 14, 512) 4608

)

conv\_dw\_11\_bn (BatchNormali (None, 14, 14, 512) 2048

zation)

conv\_dw\_11\_relu (ReLU) (None, 14, 14, 512) 0

conv\_pw\_11 (Conv2D) (None, 14, 14, 512) 262144

conv\_pw\_11\_bn (BatchNormali (None, 14, 14, 512) 2048

zation)

conv\_pw\_11\_relu (ReLU) (None, 14, 14, 512) 0

conv\_pad\_12 (ZeroPadding2D) (None, 15, 15, 512) 0

conv\_dw\_12 (DepthwiseConv2D (None, 7, 7, 512) 4608

)

conv\_dw\_12\_bn (BatchNormali (None, 7, 7, 512) 2048

zation)

conv\_dw\_12\_relu (ReLU) (None, 7, 7, 512) 0

conv\_pw\_12 (Conv2D) (None, 7, 7, 1024) 524288

conv\_pw\_12\_bn (BatchNormali (None, 7, 7, 1024) 4096

zation)

conv\_pw\_12\_relu (ReLU) (None, 7, 7, 1024) 0

conv\_dw\_13 (DepthwiseConv2D (None, 7, 7, 1024) 9216

)

conv\_dw\_13\_bn (BatchNormali (None, 7, 7, 1024) 4096

zation)

conv\_dw\_13\_relu (ReLU) (None, 7, 7, 1024) 0

conv\_pw\_13 (Conv2D) (None, 7, 7, 1024) 1048576

conv\_pw\_13\_bn (BatchNormali (None, 7, 7, 1024) 4096

zation)

conv\_pw\_13\_relu (ReLU) (None, 7, 7, 1024) 0

global\_max\_pooling2d (Globa (None, 1024) 0

lMaxPooling2D)

dropout (Dropout) (None, 1024) 0

dense (Dense) (None, 128) 131200

batch\_normalization (BatchN (None, 128) 512

ormalization)

dense\_1 (Dense) (None, 64) 8256

dropout\_1 (Dropout) (None, 64) 0

batch\_normalization\_1 (Batc (None, 64) 256

hNormalization)

dense\_2 (Dense) (None, 9) 585

=================================================================

Total params: 3,369,673

Trainable params: 3,347,401

Non-trainable params: 22,272

Details on the code …

dropout\_dense = 0.1

mobilenet\_model = MobileNet(input\_shape=IMAGE\_SHAPE, include\_top=False, pooling="max")

model = Sequential()

model.add(mobilenet\_model)

model.add(Dropout(dropout\_dense))

model.add(Dense(128, activation="relu"))

model.add(BatchNormalization())

model.add(Dense(64, activation="relu"))

model.add(Dropout(dropout\_dense))

model.add(BatchNormalization())

model.add(Dense(9, activation="softmax"))

def top\_2\_acc(y\_true, y\_pred):

    return top\_k\_categorical\_accuracy(y\_true, y\_pred, k=2)

def top\_3\_acc(y\_true, y\_pred):

    return top\_k\_categorical\_accuracy(y\_true, y\_pred, k=3)

model.compile(Adam(0.001), loss="categorical\_crossentropy", metrics=[categorical\_accuracy, top\_2\_acc, top\_3\_acc])

explain ;

IMAGE\_SHAPE is a tuple that defines the shape of the input images.

dropout\_dense is a scalar that defines the dropout probability for the dropout layers.

The MobileNet function from the Keras library is used to create a pre-trained MobileNet model that takes images of size IMAGE\_SHAPE as input, excludes the fully connected top layer, and uses global max pooling to reduce the output feature map to a vector.

The Sequential function is used to create a new sequential model.

The pre-trained MobileNet model is added to the sequential model as the first layer.

A dropout layer is added to the sequential model to randomly drop out a fraction of the units, reducing the risk of overfitting.

A fully connected dense layer with 128 units and ReLU activation is added to the sequential model.

A batch normalization layer is added to the sequential model to normalize the output of the previous layer.

Another fully connected dense layer with 64 units and ReLU activation is added to the sequential model. Another dropout layer is added to the sequential model to further reduce the risk of overfitting.

Another batch normalization layer is added to the sequential model.

A final fully connected dense layer with 9 units and softmax activation is added to the sequential model, corresponding to the 9 classes in the dataset. Two custom accuracy metrics, top\_2\_acc and top\_3\_acc, are defined using the top\_k\_categorical\_accuracy function from the Keras library.The model is compiled using the Adam optimizer with a learning rate of 0.001, categorical cross-entropy loss, and the custom accuracy metrics defined in step 13.

By using a pre-trained MobileNet model as a feature extractor, the model can effectively learn to classify the skin cancer images with a relatively small number of parameters. The dropout and batch normalization layers help to reduce the risk of overfitting, and the custom accuracy metrics measure the performance of the model in top-2 and top-3 accuracy, in addition to standard categorical accuracy.

4- Tune Hyperparameter

This is the most important part for our model in the case of learning weights , saving it

And improve the performance of the model

Code:

filepath = "model.h5"

checkpoint\_param = {

    "filepath": filepath,

    "monitor": "val\_categorical\_accuracy",

    "verbose": 1,

    "save\_best\_only": True,

    "mode": "max"

}

checkpoint = ModelCheckpoint(\*\*checkpoint\_param)

lr\_decay\_params = {

    "monitor": "val\_loss",

    "factor": 0.5,

    "patience": 2,

    "min\_lr": 1e-5

}

lr\_decay = ReduceLROnPlateau(\*\*lr\_decay\_params)

early\_stopping = EarlyStopping(monitor="val\_loss", patience=6, verbose=1)

explain:

Checkpoint : Checkpointing is a technique that involves saving the model weights during training. This can be useful in case the training process is interrupted, as it allows you to resume training from the point at which it left off. In addition, checkpointing can be used to save the best performing weights of the model, which can be useful for later use in inference or transfer learning.

Learning rate decay : Learning rate decay is a technique that involves gradually decreasing the learning rate during training. This can be useful because as the model gets closer to the optimal solution, the updates to the weights should become smaller. A decaying learning rate can help ensure that the model is making smaller and smaller updates as it gets closer to the optimal solution, preventing oscillations and helping the model converge more quickly and efficiently

Early stopping : Early stopping is a technique that involves stopping the training process when the validation loss stops improving. This can be useful because it can help prevent overfitting, which occurs when the model starts to memorize the training data instead of generalizing to new data. By stopping the training process when the validation loss stops improving, you can prevent the model from overfitting and ensure that it is generalizing well to new data

5- Fit

This is the part of fitting the model with train and validation data

print("Training the model...")

history = model.fit(train\_flow,steps\_per\_epoch=32,epochs=50,verbose=1,validation\_data=val\_flow,validation\_steps=32,callbacks = [checkpoint, lr\_decay])

print("Done!")

explain:

method of the model object and the generator objects for the training and validation datasets. Here's how the code works:

The fit method is called on the model object with the following parameters:train\_flow is the generator object for the training dataset.steps\_per\_epoch is the number of batches to be processed for each epoch. Here it is set to 32 batches. epochs is the number of times to iterate over the entire training dataset. Here it is set to 50 epochs. verbose is set to 1 to display progress bar during training. validation\_data is the generator object for the validation dataset. validation\_steps is the number of batches to be processed for each validation epoch. Here it is set to 32 batches. callbacks is a list of callbacks to be executed during training. Here, it includes the checkpoint callback to save the weights of the model with the best validation accuracy and the learning rate decay callback to reduce the learning rate if the validation loss does not improve. The model is trained on the training dataset in batches, with the weights of the pre-trained MobileNet model being frozen and only the weights of the newly added layers being updated. After each epoch, the model is evaluated on the validation dataset to monitor the performance of the model and determine if the weights should be saved. The history object is returned by the fit method, which contains the training and validation loss and accuracy values for each epoch.

The function prints "Done!" to indicate that the training process is complete.

By training the model using generator objects, the model is able to learn from a larger number of augmented examples and avoid overfitting to the training dataset. The callbacks allow for early stopping and saving the weights of the best performing model.

Here is some epochs:

Epoch 25/50

32/32 [==============================] - ETA: 0s - loss: 0.4057 - categorical\_accuracy: 0.8438 - top\_2\_acc: 0.9741 - top\_3\_acc: 0.9917

Epoch 25: val\_categorical\_accuracy improved from 0.90625 to 0.93750, saving model to model.h5

32/32 [==============================] - 33s 1s/step - loss: 0.4057 - categorical\_accuracy: 0.8438 - top\_2\_acc: 0.9741 - top\_3\_acc: 0.9917 - val\_loss: 0.3470 - val\_categorical\_accuracy: 0.9375 - val\_top\_2\_acc: 0.9375 - val\_top\_3\_acc: 0.9688 - lr: 1.2500e-04

Epoch 26/50

32/32 [==============================] - ETA: 0s - loss: 0.3911 - categorical\_accuracy: 0.8477 - top\_2\_acc: 0.9780 - top\_3\_acc: 0.9917

Epoch 26: val\_categorical\_accuracy did not improve from 0.93750

32/32 [==============================] - 32s 1s/step - loss: 0.3911 - categorical\_accuracy: 0.8477 - top\_2\_acc: 0.9780 - top\_3\_acc: 0.9917 - val\_loss: 0.5722 - val\_categorical\_accuracy: 0.8125 - val\_top\_2\_acc: 0.9062 - val\_top\_3\_acc: 0.9688 - lr: 1.2500e-04

Epoch 27/50

32/32 [==============================] - ETA: 0s - loss: 0.3718 - categorical\_accuracy: 0.8657 - top\_2\_acc: 0.9731 - top\_3\_acc: 0.9888

Epoch 27: val\_categorical\_accuracy did not improve from 0.93750

32/32 [==============================] - 31s 959ms/step - loss: 0.3718 - categorical\_accuracy: 0.8657 - top\_2\_acc: 0.9731 - top\_3\_acc: 0.9888 - val\_loss: 0.3853 - val\_categorical\_accuracy: 0.8750 - val\_top\_2\_acc: 0.9375 - val\_top\_3\_acc: 1.0000 - lr: 1.2500e-04

Epoch 28/50

32/32 [==============================] - ETA: 0s - loss: 0.3797 - categorical\_accuracy: 0.8555 - top\_2\_acc: 0.9795 - top\_3\_acc: 0.9927

Epoch 28: val\_categorical\_accuracy did not improve from 0.93750

32/32 [==============================] - 32s 994ms/step - loss: 0.3797 - categorical\_accuracy: 0.8555 - top\_2\_acc: 0.9795 - top\_3\_acc: 0.9927 - val\_loss: 0.4664 - val\_categorical\_accuracy: 0.8125 - val\_top\_2\_acc: 0.9062 - val\_top\_3\_acc: 0.9688 - lr: 6.2500e-05

Epoch 29/50

32/32 [==============================] - ETA: 0s - loss: 0.3738 - categorical\_accuracy: 0.8477 - top\_2\_acc: 0.9766 - top\_3\_acc: 0.9966

Epoch 29: val\_categorical\_accuracy did not improve from 0.93750

32/32 [==============================] - 32s 995ms/step - loss: 0.3738 - categorical\_accuracy: 0.8477 - top\_2\_acc: 0.9766 - top\_3\_acc: 0.9966 - val\_loss: 0.4994 - val\_categorical\_accuracy: 0.8438 - val\_top\_2\_acc: 0.9688 - val\_top\_3\_acc: 1.0000 - lr: 6.2500e-05

Epoch 30/50

32/32 [==============================] - ETA: 0s - loss: 0.3905 - categorical\_accuracy: 0.8477 - top\_2\_acc: 0.9756 - top\_3\_acc: 0.9941

Epoch 30: val\_categorical\_accuracy did not improve from 0.93750

32/32 [==============================] - 32s 1s/step - loss: 0.3905 - categorical\_accuracy: 0.8477 - top\_2\_acc: 0.9756 - top\_3\_acc: 0.9941 - val\_loss: 0.4271 - val\_categorical\_accuracy: 0.9062 - val\_top\_2\_acc: 0.9375 - val\_top\_3\_acc: 0.9688 - lr: 3.1250e-05

6- Model Evaluation

And as a result for the following epochs the accuracy of the model is:

\_, val\_acc, val\_top\_2\_acc, val\_top\_3\_acc = model.evaluate(val\_flow, steps=len(val\_flow))

y\_val\_true = val\_flow.classes

y\_val\_pred = np.argmax(model.predict(val\_flow, steps=len(val\_flow)), axis=1)

val\_f1\_score = f1\_score(y\_val\_true, y\_val\_pred, average="micro")

print("Validation accuracy: {:.4f}".format(val\_acc))

print("Validation top-2 accuracy: {:.4f}".format(val\_top\_2\_acc))

print("Validation top-3 accuracy: {:.4f}".format(val\_top\_3\_acc))

print("Validation F1 score: {:.4f}".format(val\_f1\_score))

output:

Validation accuracy: 0.8649

Validation top-2 accuracy: 0.9887

Validation top-3 accuracy: 0.9955

Validation F1 score: 0.8694

And as result for the testing :

\_, test\_acc, test\_top\_2\_acc, test\_top\_3\_acc = model.evaluate(test\_flow, steps=len(test\_flow))

y\_test\_true = test\_flow.classes

y\_test\_pred = np.argmax(model.predict(test\_flow, steps=len(test\_flow)), axis=1)

test\_f1\_score = f1\_score(y\_test\_true, y\_test\_pred, average="micro")

print("Test accuracy: {:.4f}".format(test\_acc))

print("Test top-2 accuracy: {:.4f}".format(test\_top\_2\_acc))

print("Test top-3 accuracy: {:.4f}".format(test\_top\_3\_acc))

print("Test F1 score: {:.4f}".format(test\_f1\_score))

output:

Test accuracy: 0.5847

Test top-2 accuracy: 0.8136

Test top-3 accuracy: 0.8559

Test F1 score: 0.5847

7- charts of Losses and Accuracy

And there is the part of Visualization the accuracy and losses

loss\_train = history.history["loss"]

acc\_train = history.history["categorical\_accuracy"]

loss\_val = history.history["val\_loss"]

acc\_val = history.history["val\_categorical\_accuracy"]

epochs = np.arange(1, len(loss\_train) + 1)

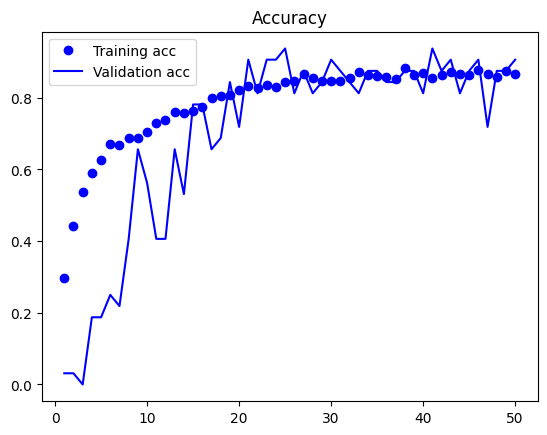
plt.plot(epochs, acc\_train, "bo", label="Training acc")

plt.plot(epochs, acc\_val, "b", label="Validation acc")

plt.title("Accuracy")

plt.legend()

plt.show()



And from the following chart we see the result of the model as we wanted it to be

Each validation and training accuracy is close together from the chart

And here the part of the loss

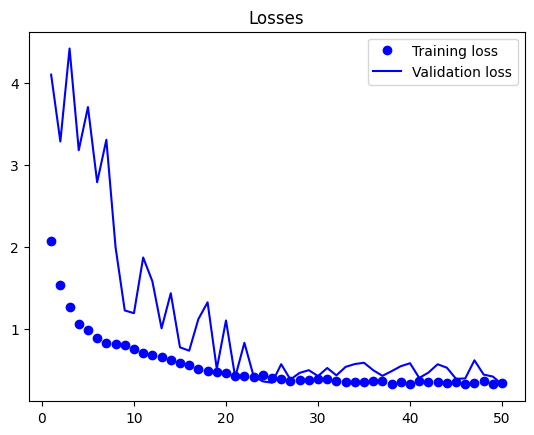
plt.plot(epochs, loss\_train, "bo", label="Training loss")

plt.plot(epochs, loss\_val, "b", label="Validation loss")

plt.title("Losses")

plt.legend()

plt.show()



From the following chart the loss is decreasing through the epochs for the training and validation Losses

8- Confusion matrix

\* some specific benefits of using a confusion matrix:

\* Evaluation of model accuracy

\* Identification of errors

\* Comparison of model performance

\* Adjusting the model

\* Communication with stakeholders

Code :

conf\_mat = confusion\_matrix(y\_val\_true, y\_val\_pred) plt.imshow(conf\_mat, interpolation="nearest", cmap=plt.cm.Blues)

plt.title("Confusion matrix")

plt.colorbar()

tick\_marks = np.arange(len(class\_names))

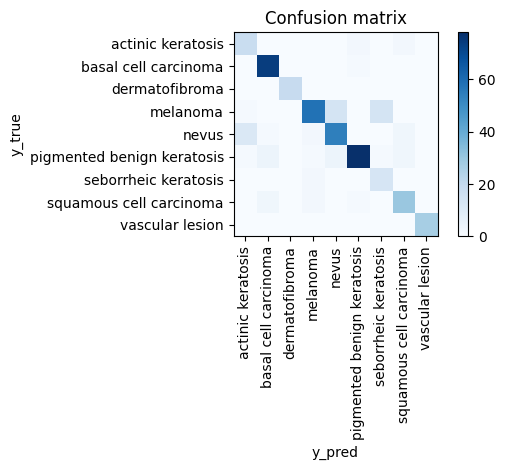
plt.xticks(tick\_marks, class\_names, rotation=85)

plt.yticks(tick\_marks, class\_names)

plt.ylabel("y\_true")

plt.xlabel("y\_pred")

plt.tight\_layout()



9 - ***Evaluation Metrics***

Accuracy = *(*TP + TN*) / (*TP + TN + FP + FN*) (1)*

Precision = TP / *(*TP + FP*) (2)*

Recall = TP / *(*TP + FN*)* (3)

F1-score = 2 \* ( Precision ∗ Recall) / ( Precision + Recall) (4)

Code:

import numpy as np

from sklearn.metrics import classification\_report

# Make predictions on the test set

y\_pred = model.predict(test\_flow)

# Convert predictions to class labels

y\_pred = np.argmax(y\_pred, axis=1)

# Convert true labels to class labels

y\_true = test\_flow.classes

# Calculate precision, recall, and f1 score

report = classification\_report(y\_true, y\_pred, target\_names = class\_names)

print(report)

precision recall f1-score support

actinic keratosis 1.00 0.31 0.48 16

basal cell carcinoma 0.62 0.81 0.70 16

dermatofibroma 0.89 0.50 0.64 16

melanoma 0.00 0.00 0.00 16

nevus 0.39 0.94 0.56 16

pigmented benign keratosis 0.46 0.69 0.55 16

seborrheic keratosis 0.00 0.00 0.00 3

squamous cell carcinoma 0.50 0.31 0.38 16

vascular lesion 1.00 1.00 1.00 3

accuracy 0.51 118

macro avg 0.54 0.51 0.48 118

weighted avg 0.55 0.51 0.47 118

# Make predictions on the test set

y\_pred = model.predict(val\_flow)

# Convert predictions to class labels

y\_pred = np.argmax(y\_pred, axis=1)

# Convert true labels to class labels

y\_true = val\_flow.classes

# Calculate precision, recall, and f1 score

report = classification\_report(y\_true, y\_pred, target\_names = class\_names)

print(report)

output:

precision recall f1-score support

actinic keratosis 0.61 0.86 0.72 22

basal cell carcinoma 0.88 0.97 0.92 75

dermatofibroma 1.00 1.00 1.00 19

melanoma 0.92 0.76 0.83 87

nevus 0.85 0.75 0.80 71

pigmented benign keratosis 0.91 0.86 0.88 92

seborrheic keratosis 0.46 0.73 0.56 15

squamous cell carcinoma 0.84 0.89 0.86 36

vascular lesion 0.96 1.00 0.98 27

accuracy 0.85 444

macro avg 0.83 0.87 0.84 444

weighted avg 0.87 0.85 0.86 444

Faculty of computer science and information

Selected topics in computer science–2 (cs\_396)

|  |  |
| --- | --- |
| **Member​** | **ID​** |
| **دنيا سيد محمد محمدي**​ | **202000297**​ |
| **أحمد مصطفي محمد زين الدين**​ | **202000088**​ |
| **أحمد محي عبد الحي بيومي**​ | **202000085**​ |
| **تسنيم حاتم وفيق العدوي**​ | **202000222**​ |
| **حورية محمود عبد العليم**​ | **202000282**​ |
| **عمر عماد الدين محمد**​ | **202000604**​ |

References:

\* <https://arxiv.org/abs/1909.09586>

\* <https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>

\* <https://arxiv.org/abs/1912.05911>

\* <https://arxiv.org/abs/1910.05446>

\* <https://arxiv.org/abs/2003.05689>

\* <https://arxiv.org/abs/1811.12808>

\* <https://www.researchgate.net/publication/355096788_Confusion_Matrix>

\* <https://www.researchgate.net/figure/A-Epoch-vs-sparse-categorical-cross-entropy-loss-function_fig2_352705128>

\* <https://paperswithcode.com/method/softmax>

\* <https://youtu.be/ySEx_Bqxvvo>

\* <https://youtu.be/_JB0AO7QxSA>

\* <https://youtu.be/wEoyxE0GP2M>

\*Mobile Net V1: <https://arxiv.org/abs/1704.04861>

\*Mobile Net V2: <https://arxiv.org/abs/1801.04381>

\*Mobile Net V3: <https://arxiv.org/abs/1905.02244>

\*MobileNetV3-Small: <https://arxiv.org/abs/1905.05908>

\*MobileNetV3-Large: <https://arxiv.org/abs/1905.02244>

\* Stewart, B.W., Wild, C.: International Agency for Research on Cancer, and World Health

Organization. World cancer report (2014)

\* Cakir, B.O.,Adamson, P., Cingi, C.: Epidemiology and economic burden of nonmelanoma skin

cancer. Facial Plast. Surg. Clin. North Am. **20**(4), 419–422 (2012). https://doi.org/10.1016/j.

fsc.2012.07.004

\* Rogers, H.W., Weinstock, M.A., Feldman, S.R., Coldiron, B.M.: Incidence estimate of

nonmelanoma skin cancer (Keratinocyte carcinomas) in the U.S. population, 2012. JAMA

Dermatol. **151**(10), 1081–1086 (2015). <https://doi.org/10.1001/jamadermatol.2015.1187>

\* Stern, R.S.: Prevalence of a history of skin cancer in 2.007: results of an incidence-basedmodel.

Arch. Dermatol. **146**(3), 279–282 (2010). https://doi.org/10.1001/archdermatol.2010.4

\* WHO: Skin cancers WHO (2017)

\* Koh, H.K.,Geller, A.C.,Miller, D.R.,Grossbart,T.A.,Lew,R.A.: Prevention and early detection

strategies for melanoma and skin cancer current status. Arch. Dermatol. **132**(4), 436–443

(1996). <https://doi.org/10.1001/archderm.1996.03890280098014>

\* Parkin, D.M., Mesher, D., Sasieni, P.: Cancers attributable to solar (ultraviolet) radiation exposure

in the UK in 2010. Br. J. Cancer **105**(2), S66–S69 (2011). https://doi.org/10.1038/bjc.

2011.486

\* Canadian Cancer Society. Risk factors for melanoma skin cancer (2018). https://www.cancer.

org/cancer/melanoma-skin-cancer/causes-risks-prevention/risk-factors.html.Accessed 31Mar

2019

\*Khosla, A., et al.: ImageNet large scale visual recognition challenge. Int. J. Comput. Vis.

**115**(3), 211–252 (2015). <https://doi.org/10.1007/s11263-015-0816-y>

\*Rosado, B., et al.: Accuracy of computer diagnosis of melanoma: a quantitative meta-analysis.

Arch. Dermatol. **139**(3), 361–367 (2003). <https://doi.org/10.1001/archderm.139.3.361>

\*Burroni,M., et al.: Melanoma computer-aided diagnosis: reliability and feasibility study. Clin.

Cancer Res. **10**(6), 1881–1886 (2004). <https://doi.org/10.1158/1078-0432.CCR-03-0039>

\*Kawahara, J., Hamarneh,G.: Multi-resolution-tractCNNwith hybrid pretrained and skin-lesion

trained layers. In:Wang, L., Adeli, E.,Wang, Q., Shi, Y., Suk, H.I. (eds.) Machine Learning in

Medical Imaging. MLMI 2016. Lecture Notes in Computer Science, p. 10019 (2016). https://

doi.org/10.1007/978-3-319-47157-0\_20

\*Milton, M.A.A.: Automated Skin Lesion Classification Using Ensemble of Deep Neural

Networks in ISIC 2018: Skin Lesion Analysis TowardsMelanoma Detection Challenge (2019)

\*Hardie, R.C., Ali, R., De Silva, M.S., Kebede, T.M.: Skin Lesion Segmentation and Classification

for ISIC 2018 Using Traditional Classifiers with Hand-Crafted Features. https://arxiv.

org/abs/1807.07001

\*Howard, A.G., et al.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision

Applications (2017). <https://arxiv.org/abs/1704.04861>

\* Tschandl, P., Rosendahl, C., Kittler, H.: The HAM10000 dataset, a large collection of multisources

dermatoscopic images of common pigmented skin lesions. Sci. Data **5**, 180161 (2018).

<https://doi.org/10.1038/sdata.2018.161>

\*Weiss, K., Khoshgoftaar, T.M., Wang, D.: A survey of transfer learning. J. Big Data **3**(1),

1345–1359 (2016). <https://doi.org/10.1109/TKDE.2009.191>

\*Image Preprocessing—Keras Documentation. Keras (2019). https://keras.io/preprocessing/

image/. Accessed 31 Mar 2019

\* Pandas: Working with missing data—pandas 0.22.0 documentation (2019). https://pandas.

pydata.org/pandas-docs/stable/user\_guide/missing\_data.html. Accessed 31 Mar 2019

\*Mikolajczyk, A., Grochowski, M.: Data augmentation for improving deep learning in image

classification problem. In: International Interdisciplinary Ph.D.Workshop (IIPhDW), pp. 117–

122 (2018). https://doi.org/10.1109/IIPHDW.2018.8388338

43. Kaggle: Your Home for Data Science. https://www.kaggle.com/. Accessed 31 Mar 2019