PROJECT REPORT

COMPARATIVE ANALYSIS OF EFFICIENTNET AND RESNET MODELS IN THE CLASSIFICATION OF SKIN CANCER

DANIEL HARTANTO
20.K1.0008

Faculty of Computer Science
Soegijapranata Catholic University
2024

APPROVAL AND RATIFICATION PAGE

Judul Tugas Akhir	:	Comparative Analysis of EfficientNet and ResNet Models in The Classification of Skin Cancer
Diajukan oleh	:	DANIEL HARTANTO
NIM	:	20.K1.0008
Tanggal disetujui	:	08 Januari 2024
Telah setujui oleh		
Pembimbing	:	Rosita Herawati S.T., M.I.T.
Penguji 1	:	Yonathan Purbo Santosa S.Kom., M.Sc.
Penguji 2	:	Hironimus Leong S.Kom., M.Kom.
Penguji 3	:	R. Setiawan Aji Nugroho S.T., MCompIT., Ph.D
Penguji 4	:	Rosita Herawati S.T., M.I.T.
Penguji 5	:	Y.b. Dwi Setianto S.T., M.Cs.
Penguji 6	:	Dr. Yulianto Tejo Putranto S.T., M.T.
Ketua Program Studi	:	Rosita Herawati S.T., M.I.T.
Dekan	:	Prof. Dr. F. Ridwan Sanjaya S.E., S.Kom., MS.IEC.

Halaman ini merupakan halaman yang sah dan dapat diverifikasi melalui alamat di bawah ini.

sintak.unika.ac.id/skripsi/verifikasi/?id=20.K1.0008

DECLARATION OF AUTHORSHIP

I, the undersigned:

Name : Daniel Hartanto

ID : 20.K1.0008

declare that this work, titled " COMPARATIVE ANALYSIS OF EFFICIENTNET AND RESNET MODELS IN THE CLASSIFICATION OF SKIN CANCER", and the work presented in it is my own. I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at Soegijapranata Catholic University
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
3. Where I have consulted the published work of others, this is always clearly attributed.
4. Where I have quoted from the work of others, the source is always given.
5. Except for such quotations, this work is entirely my own work.
6. I have acknowledged all main sources of help.
7. Where the work is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Semarang, 8 January 2024

Daniel Hartanto

20.K1.0008

HALAMAN PERNYATAAN PUBLIKASI KARYA ILMIAH UNTUK KEPENTINGAN AKADEMIS

Yang bertanda tangan dibawah ini:

Nama	: Daniel Hartanto
Program Studi	: Teknik Informatika
Fakultas	: Ilmu Komputer
Jenis Karya	: Skripsi

Menyetujui untuk memberikan kepada Universitas Katolik Soegijapranata Semarang Hak Bebas Royalti Nonekslusif atas karya ilmiah yang berjudul " Comparative Analysis of EfficientNet and ResNet Models in The Classification of Skin Cancer ". Dengan Hak Bebas Royalti Nonekslusif ini Universitas Katolik Soegijapranata berhak menyimpan, mengalihkan media/formatkan, mengelola dalam bentuk pangkalan data (database), merawat, dan mempublikasikan tugas akhir ini selama tetap mencantumkan nama saya sebagai penulis / pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Semarang, 8 Januari 2024
Yang menyatakan

Daniel Hartanto
20.K1.0008

ACKNOWLEDGMENT

I have received a myriad of support, advice, and assistance throughout this document writing. I would like to thank my supervisors Rosita Herawati S.T., M.I.T. in helping me in this project. I am also grateful to other lecturers who have guided me during my studies which I cannot mention by name. I would also like to thank my friend for guiding with advice to finish this document. Their advice were a source of motivation, pushing me towards the successful completion of this project. Then thank you to my family who giving me support to motivating me to do the project. I also would like to thank my family and friends for giving me ceaseless love, support, and advices throughout my study in Soegijapranata Catholic University. I am very grateful to all of them because this project would not have been completed without their support and advice.

ABSTRACT

Skin cancer get classified as one of the most common types of cancer cause to death. There are some types of skin cancer as: basal cell carcinoma (BCC), melanoma (MEL), and others. This cancer may have different symptoms depending on the type of skin cancer, but the most common signs include changes in the size, shape, or color of a mole or skin. The progress in machine learning has been increasing, mainly on deep learning and artificial intelegenct. In the recent past deep learning has been developed for medical research. In the latest papers, algorithms that have been applied for medical research are pre-trained models. In this research, the author compares the pre-trained EffecientNet and ResNet-50 for classification of skin cancer on the HAM10000 dataset to find out which is the best for classifying skin cancer and what is the best pre-trained model for skin cancer classification. This study aims to find the pre-trained EffecientNet and ResNet-50 models for accurate and efficient for skin cancer classification. In this experiment the results obtained were: that the highest accuracy on test was achieved by EfficientNet B7 on 88.41% accuracy and the lowest accuracy on test was achieved by ResNet 50 on 83.42% accuracy.

Keyword: skin cancer, Pre-trained, EfficientNet, ResNet-50

TABLE OF CONTENTS

LIST OF FIGURE

LIST OF TABLE

CHAPTER 1
INTRODUCTION

1.1. Background

Skin cancer is one of the most common cancers that cause death. Skin cancer can be categorized into several types like basal cell carcinoma (BCC), melanoma (MEL), and many others. Symptoms of skin cancer can vary depending on the type of cancer, but the most common signs include changes in the size, shape, or color of a mole or skin. Skin cancer is a serious health concern that needs to be prevented from the beginning and treated with proper awareness and care.

Regular skin self-check is crucial for detecting early-stage skin cancer. Therefore, it is vital to examine any changes in moles or skin conditions that can alert individuals to potential warning signs, allowing them to intervene early. Healthcare professionals can further enhance the diagnosis process by including a supporting image technique. This method enables accurate classification of suspicious skin lesions using image classification methods. Because of this, timely detection and intervention can occur, leading to an increased chance of successfully identifying and treating skin cancer during its earliest stages.

In recent, the progress of machine learning mainly in deep learning and artificial intelligence has been increasing especially in medical research. Two algorithms that have recently been applied for medical research are the pre-trained CNN. Pre-trained CNN is an algorithm that uses a CNN that has been trained on a large dataset. Overall, the progress of machine learning, especially in deep learning and artificial intelligence, has greatly contributed to improving medical research.

In this study, the author HAM10000 dataset contains about 7 types of pigmented lesions. The author compares pre-trained CNN EffecientNet and ResNet-50 to predict and classify skin cancer on the HAM10000 dataset to find the best accuracy. By evaluating and comparing the performance of different algorithms, the author can determine which algorithm is best used for skin cancer classification.

.

1.2. Problem Formulation

From the background above, the problem can be formulated as following:

1. How does the performance of Pre-trained EffecientNet and Pre-trained ResNet-50 classify skin cancer?
2. Which the best algorithm to classify skin cancer?

1.3. Scope

This study only compares the accuracy between EfficientNet and ResNet-50 algorithms without considering the speed between algorithms. The dataset used is a dataset from Kaggle, namely Skin Cancer: HAM10000, which contains around 10000 pictures. This study classifies the dataset into 7 classes. This study splits the data into 80% training, 10% testing, and 10% validation. In this study, pre-processing will be used to remove the noise from the picture. The number of epochs that will be used as a parameter is around 50 epochs to analyze its impact on the accuracy comparison between the two algorithms.

1.4. Objective

The objective of this study is to find which one is the most effective between pre-trained CNN EffecientNet and ResNet-50 on classifying skin cancer. In addition, this study aims to identify the optimal parameters for EfficientNet to classify skin cancer. The same goes for ResNet to find the optimal parameters for ResNet-50 to classify skin cancer. By comparing these models and fine-tuning parameters, this study seeks which model and parameter to get the best accuracy in skin cancer classification.

CHAPTER 2
LITERATURE STUDY

Based on Huang et al.[1] who developed a binary classification and multiclass classification model on Kaohsiung Chang Gung Memorial Hospital (KCGMH) and on the HAM10000 dataset. The methods that are used in this research are EfficientNet and DenseNet. The Kaohsiung Chang Gung Memorial Hospital (KCGMH) Kaohsiung Chang Gung Memorial Hospital (KCGMH) dataset only contains five classes with total 1278 images and the HAM10000 dataset contains 7 classes with total 10015 images. The images in this research were cropped into 224 x 224 pixel and 112 x 122 pixel. The research said that DenseNet-121 has a remarkable performance in binary classification on the KCGMH dataset with 89,5% accuracy. Still, EfficientNet B-4 has 85,8% accuracy for the seven-class classification HAM10000 and 72,1% accuracy for the KCGMH dataset multiclass classification. for multiclass classification the best is EfficientNet.

On the other hand, classifying multiclass classification using EfficientNet was done by Ali et al.[2]. This research was performed using transfer learning on pre trained imagenet weights. The experiment is an unbalanced HAM10000 dataset to classify skin cancer. The researcher removed hairs from the image because the researcher said that if the noise doesn't get removed the CNN will need to learn to ignore the noise. this research uses the unbalanced dataset and balancing the dataset with the augmented dataset. The augmented that used in the dataset are using rotation, zoom, vertical flip and horizontal flip. The best performance on EfficientNets B4 and B5 is 88% on precision, 88% on recall, 87% on F1 score, Specificity of 88 percent, and the area under the receiver operating characteristics(Roc Auc) Score of 97.5 percent. the F1 Score of the best model EfficientNet B4.

There was research that talked about creating a proposed model using HAM10000 dataset. According to Islam et al.[3], using their proposed model to classify skin cancer into 2 classes which are benign cancer and malignant cancer. The dataset contains around 10015 data of images but there are three labeled skin lesions which are benign, malignant, and unknown. But in the research only using benign and malignant so the dataset that is used is 6705 benign images and 2135 malignant images. The preprocessing that is used is enhancing images and resizing the images.

Besides that, the research also uses data augmentation to gain a good classification. From the proposed model the accuracy is 96.10% in training and 90.63% in testing.

The proposed method used by Tahir et al.[4] is a deep learning-based skin cancer classification network (DSCC_Net). The datasets that are used are ISIC 2020, HAM10000, and DermIS. To resolve the unbalanced dataset the research was using an up-sampling algorithm named the synthetic minority oversampling technique (SMOTE) Tomek. The study also compares their proposed method with other CNN models such as VGG-19, ResNet-152, VGG-16, MobileNet, Inception-V3, and EfficientNet B0. The image in the research was resized into 150 x 150 pixels. The research was trained with only four categories which are SCC, BCC, MN, and MEL. The proposed method gets 99.43% AUC, 94.17% accuracy, 93.76% recall, 94.28% precision, and 93.93% F1 score.

There was a research that compares a proposed deep convolutional neural network (DCNN) method and compared with some transfer learning models like AlexNet, ResNet, VGG-16, DenseNet, and MobileNet, the research used to HAM10000 dataset to classify benign and malignant skin. The processes that were used were removing the noise, normalizing input, and augmenting the image. The study said that the unbalanced dataset making the model doesn't get a better accuracy. to increase the accuracy the research was using augmentationq. The research gets a better classification rate compared with other transfer learning models. Ali et al.[5] get 93,16% of training and 91,93 testing accuracy.

Popescu et al.[6] using the HAM10000 dataset to train the data to predict seven types of skin lesions. The research combined nine types of models which are AlexNet, GoogLeNet, GoogLeNet-Places365, MobileNet-V2, Xception, ResNet-50, ResNet-101, InceptionResNet-V2, and DenseNet201. With the weight matrix, the new matrix is used to build a multi-network ensemble system by combining each neural network. The research said that there is no risk of overfitting because of the multi-network system by considering each output. The accuracy of combining the model was 86.71%.

Khamparia et al.[7] classify skin cancer with binary classification (benign and malignant) using a framework using pre-trained CNN architectures such as Inception V3, VGG19, SqueezeNet, and Resnet50 from the International Skin Imaging Collaboration (ISIC) image archive dataset. from 5000 pictures taken, the data split into 70%-30% and 80%-20%. The average accuracy from the framework that was created is 99,2% for 80%-20% data splitting and 99,6% for 70%-30% data splitting. the framework gets more high accuracy than the training alone. From the result, using pre-trained alone is high but using it together can have a higher accuracy. From the experiment, the highest score after the proposed framework is ResNet50.

This study compares ResNet50 and MobileNet to classify multiclass classification. The dataset used is the HAM10000 dataset. using 9077 images for training and 938 images for validation. The type of hyper-tuning parameters used are optimizer, dropout rate, learning rate, and epochs. The optimizer used is Adam. The dropout rate used for Resnet is 0.4 and the dropout rate used for MobileNet is 0.25. The learning rate used is 0.001. and the epoch is 100. The results from the experiment are from the training ResNet50 has better best epoch validation loss, best epoch top 2 accuracies, better categorical accuracy, and higher accuracy 83% than MobileNet 72% accuracy. Mohapatra et al.[8] conclude that MobileNet performs better than ResNet50 when performing binary classification for cancerous and non-cancerous images specifically while ResNet50 performs better than MobileNet when it is classifying an image into more than two classes.

The dataset used is HAM10000. Because the HAM10000 contains various types of skin cancer the research uses only two cancer types and one non-cancer type. Because the types are very unbalanced, augmented is performed. If the data is unbalanced it can be becoming biased. The augmented method includes crop, scaling, contrast, brightest adjustment, horizontal flip, and vertical flip so the data becomes 1000 of each type. After that, the image was resized to 64 x 64 pixels. Abuared et al.[9] got the result 0.985 on training accuracy, and 0.975 on test accuracy.

The experiment uses data HAM10000 dataset. The dataset was split into 80% training and 20% testing. The dataset was preprocessed including resizing, augmentation, and labeling. The size images are resized into 240 x 240 pixels. The method that is used is EfficientNet B1. The augmentation was randomly between width shift +-20% and height shift +- 20% and random max 0.2-degree shear angle in a counterclockwise direction. Tajerian et al.[10] achieved 84.3% on accuracy. for the F1-score the best was 0.93 for melanocytic nevi, followed by Actinic Keratosis 0.63, Basal Cell Carcinoma 0.72, Benign Keratosis 0.70, Dermatofibroma 0.54, Melanoma 0.58, and Vascular lesions 0.80.

Based on previous research about skin cancer classification that has been done before. Huang et al.[1] said that the best method for multiclass classification is EfficientNet and Mohapatra et al.[8] said that the best method for multiclass classification is ResNet-50. The dataset was used by Mohapatra et al.[8] which is the HAM10000 dataset. In this research, the author wants to compare pre-trained CNN using EfficientNet and Resnet-50 on the HAM10000 dataset.

CHAPTER 3
RESEARCH METHODOLOGY

3.1. Research Methodology

To achieve similar results in this research study, it is important to clearly define the structured research methods clearly. If the research method is not explained in detail, the output will vary differently compared to this research because even when using the same method and the same dataset, the result can be very different. The following steps to increasing the probability of achieving similar results between studies demand implementation of these steps:

1. Literature study related to the topic in the project.
2. Collecting datasets, learning the algorithm used.
3. Preprocessing dataset and augmenting the data.
4. Implementation using EfficientNet and ResNet50
5. Analyze results of implementation and make conclusion.

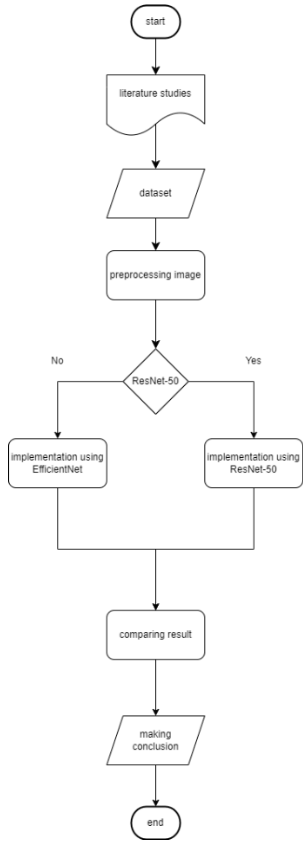


Figure 3.1 Research Methodology

3.2. Dataset Collection

In this research the dataset that will be used was taken from Kaggle, namely Skin Cancer: HAM10000 with a total size of 3 GB. The dataset contains one CSV file, 10015 images, and its mask. The CSV file contains the image name and the ground truth of its class. The images are contained 1113 images of melanoma (MEL), 6705 images of melanocytic nevi (NV), 514 images of basal cell carcinoma (BCC), 327 images of Actinic keratoses and intraepithelial carcinoma / Bowen's disease (AKIEC), 1099 images of benign keratosis-like lesions (BKL), 115 images of dermatofibroma (DF), and 142 images of vascular lesions (VASC).

3.3. Data Preprocessing

To get the best result, it is important to clean the data before into the model. Because models such as ResNet50 and EfficientNetB0 need an image of the size 224x224 pixels but the image itself is 600x450 pixels the author needs to resize it into 224x224 pixels. And on the image dataset namely HAM10000 there is an image of skin cancer but inside the image, there is some noise that covers the image which is hair. So, the author wants to remove the hair from the image. There are several step to remove the hair from the image:

1. Changing the image into grayscale.
2. Making the matrix with size 9x9 for defining the neighborhood for the morphological operation
3. Using morphologyEx to highlight the darker image by using MORPH_BLACKHAT
4. Using gausian blur to remove the noise and smoothing the noise that need to removed from the morphologyEx
5. Using the threshold to get the masking value which is the noise or the hair from the image
6. Using inpaint to replace the unwantted noise in this case is hair from the image

3.4. Models

In this project, after preprocessing the data, splitting the data was important because training data are things that will be trained with the model to predict so it will be accurate. In this project, the author split the data into 3 which are training, validation, and testing with 80% training data, 10% validation data, and 10% testing data. After splitting the data, the training data can be used to model classifications. In this project pretrained model was used. The difference between pretrained and without pretrained is the model on pretrained are already trained with a large number of data but the model without pretrained is not trained with data so it need a large of data to train the model. Because there are not enough skin cancer image for several classes pretrained model was used because HAM 10000 dataset doesn’t have enough dataset for several classes like dermatofibroma (DF) only contain 115 images . In this project, author use two different models namely EfficientNet and ResNet-50.

3.4.1. EfficientNet

The EfficientNet is a type of convolutional neural network architecture that employs a scaling method to uniformly scale its depth, width, and resolution dimensions using compound scaling. Compound scaling means if the input image is bigger it means that the network also needs more layers. This stands in contrast to conventional practices that can arbitrarily scale these factors, utilizing specific scaling coefficients instead ensures the scaling across network width, depth, and resolution remains consistent and uniform. There are some types of EfficientNet which are EfficientNet B0, EfficientNet B1, EfficientNet B2, EfficientNet B3, EfficientNet B4, EfficientNet B5, EfficientNet B6, and EfficientNet B7.

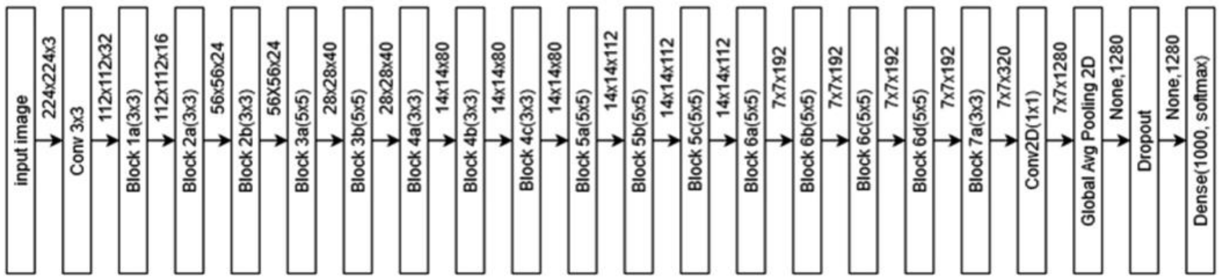


Figure 3.2 EfficientNet B0 Block Diagram[2]

3.4.2. ResNet-50

A residual Network usually called ResNet is one of the deep-learning models used for image recognition. ResNet is a Convolutional Neural Network (CNN) that supports up to a hundred layers. This method is known for its skip connection with residual block. With residual block, ResNet can skip layers about 2-3 layers at one time. ResNet has a lot of types of architecture by its layer such as ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-150. This experiment uses ResNet 50 as its model. Resnet 50 contains one MaxPool layer, one average pool layer, and forty-eight convolutional layers.

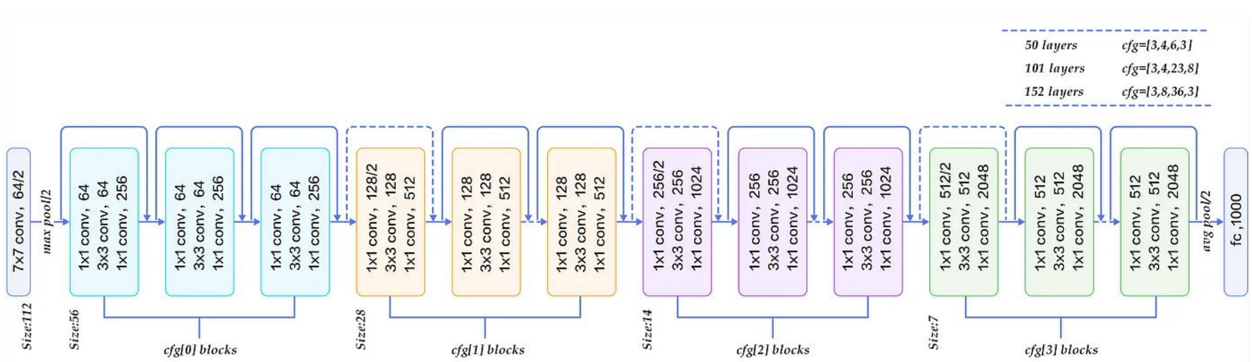


Figure 3.3 ResNet-50 Architecture[11]

3.5. Result Analysis

This study will compare the Resnet50 and EfficientNet B0-B7 with the accuracy of it The author will record the result of training accuracy, validation accuracy, and test accuracy using EfficientNet and ResNet-50. After all results have been recorded comparing the result between EfficientNet and ResNet-50 the author will be made to be able to see the comparative value between the two algorithm models between EfficientNet and ResNet-50. By comparing between two algorithm models EfficientNet and ResNet-50 it can understand which model is better and more efficient in classifying skin cancer.

CHAPTER 4
IMPLEMENTATION AND RESULTS

4.1. Experiment Setup

This research was conducted using a laptop with the following specifications: Intel I7-7700HQ, 16 GB of RAM, and NVIDIA GeForce GTX 1050. The author uses Python as a programming language. To run the code, the author uses a Kaggle notebook to run the experiment. Kaggle Notebook provides a GPU that can run the code much faster. The GPU that the author used for this experiment is GPU P1000 from the Kaggle Notebook.

4.2. Implementation

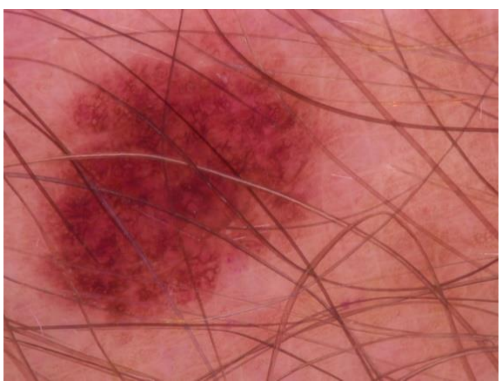
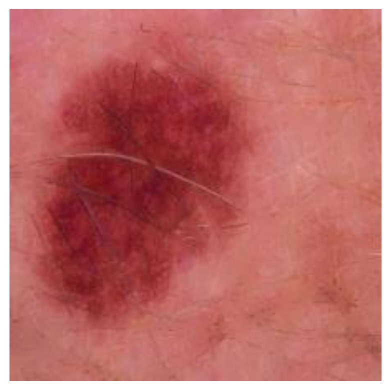
First, it is important to import some libraries that need to be used to run code. After importing the important library, the next important one is collecting the dataset. The author acquired the dataset from Kaggle namely Skin cancer: HAM10000 with size 3 GB. After downloading the dataset the author needs to read the CSV that contains the file name and the class because the list of the class is from the CSV only. The CSV contains the names of the images and their ground truth. To read the CSV the author used Pandas library. After reading the CSV the author moves the image into its class type one by one. When the author moves the image, the author also preprocesses the image. The preprocessing that the author uses is removing the hair from the images.

```
classes=df.columns[1:]

def preprocessing(image):
    gambar=cv2.imread(f"/kaggle/input/ham1000-segmentation-and-classification/images/{image}.jpg",cv2.IMREAD_COLOR)
    resize=cv2.resize(gambar,[224,224])
    grayScale = cv2.cvtColor(resize, cv2.COLOR_RGB2GRAY )
    kernel = cv2.getStructuringElement(1,(9,9))
    blackhat = cv2.morphologyEx(grayScale, cv2.MORPH_BLACKHAT, kernel)
    bhg= cv2.GaussianBlur(blackhat,(3,3),cv2.BORDER_DEFAULT)
    ret,mask = cv2.threshold(bhg,10,255,cv2.THRESH_BINARY)
    dst = cv2.inpaint(resize,mask,6,cv2.INPAINT_TELEA)
    return dst
for cls in classes:
    images = df[df[cls]==1]['image'].to_list()
    for image in images:
        gambar=preprocessing(image)
        cv2.imwrite(f"baru/{cls}/{image}.jpg",gambar)
```

Line 11 is used to detect all classes that are available. Lines 2-12 are used to preprocess the image to resize the image and remove the hair from the images. Line 6 is changing the images into the grayscale by use cv2.cvtColor. Line 7 has been used to make kernel to be processed on line 8. Cv2. getStructuringElement is used to create a structuring element that is used as a kernel in morphological operation. Line 8 performs its hair with morphologyEx to highlight the dark region by taking kernel that formed at line 7. Cv2.morphologyEx is the basic transformation uses input and kernel. Cv2.MORPH_BLACKHAT is the difference between the closing input image and the image. Blackhat is used for increasing the dark region in the images .Line 9 is used to remove the noise and smoothing the noise that need to removed. Line 10-11 is used to mask the difference and remove the hair from the original images. Line 13-17 is to detect the image file name that is in the class then preprocess it and move it into each class. There are some sample of the preprocessing:

Table 4.1. Sample of Preprocessing

Before Preprocessing	**After Preprocessing**
	

After preprocessing and moving the image into each class. The author reads the data by listing the directory and reading the image file to save the location and the type of images. After reading the image file and typing the type combine it becoming one table that contains the file location of the image and the type. Then the author splits the data into train, validation, and test.

```
18.  def splitting(location):
19.      files=[]
20.      labels=[]
21.      classes=os.listdir(location)
22.      for cls in classes:
23.          path=os.path.join(location,cls)
24.          list_file=os.listdir(path)
25.          for file in list_file:
```

```
26.                file_path=os.path.join(path,file)
27.                files.append(file_path)
28.                labels.append(cls)
29.        filepath=pd.Series(files,name="path")
30.        labelpath=pd.Series(labels,name="label")
31.        dataset=pd.concat([filepath,labelpath],axis=1)
32.        strat=dataset['label']
33.
   train_dataset,test_valid_dataset=train_test_split(dataset,train_size=0.8,s
   huffle=True,random_state=42,stratify=strat)
34.        strat_test_valid=test_valid_dataset["label"]
35.
   test_dataset,validation_dataset=train_test_split(test_valid_dataset,train_
   size=0.5,shuffle=True,random_state=42,stratify=strat_test_valid)
36.        print('train_df length: ', len(train_dataset), '   test_df length:
   ',len(test_dataset), '  valid_df length: ', len(validation_dataset))
37.        return train_dataset,test_dataset,validation_dataset
```

Line 18-37 are used to split the data into train, validation, and test datasets. Lines 19-31are used to make tables that contain the file image location and its label. The author uses the table that contains file image location because the author already separates the image of each class, so the author uses looping to detect the image inside each class and append the location and the label. After the table that contains file image location and label, on lines 49-53 splitting the data into 80% train, 10% validation, and 10% split.

```
38.  learning_rate=1e-4
39.  batch_size=32
40.  img_size=(224,224)
41.  img_shape=(224,224,3)
42.  epoch=30
43.  factor=learning_rate/epoch
44.  def resnet_model1():
45.      input=Input(shape=img_shape)
46.      base= tf.keras.applications.resnet_v2.ResNet50V2 (weights='
   /kaggle/input/resnetv2-
   weight/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5’
   ,include_top=False,input_shape=img_shape,input_tensor=input,classes=7)
47.
48.  #     x=Flatten()(base.output)
49.      x = GlobalAveragePooling2D()(base.output)
50.      x = BatchNormalization()(x)
51.  #
52.  #     x = Dropout(0.2)(x)
53.      x = Dropout(0.5)(x)
54.  #     x = Dense(512, activation='relu')(x)
55.  #     x = Dense(256, activation='relu')(x)
56.  #     x = Dropout(0.5)(x)
57.      output = Dense(7, activation='softmax', kernel_regularizer=
   regularizers.L1L2(l1=0.01, l2=0.01))(x)
58.      optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
59.      model.compile(optimizer=optimizer, loss="categorical_crossentropy",
   metrics=['accuracy'])
```

```
60.        return model
61.   resnet_model=resnet_model1()
62.   reduce_lr =tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
   factor=factor,patience=5, min_lr=learning_rate,verbose=1)
63.   checkpoint=tf.keras.callbacks.ModelCheckpoint("resnet_model.h5",monitor=
   "val_loss",mode="min",save_best_only = True,verbose=1)
64.   callback_list = [ reduce_lr, checkpoint]
65.   classifier_history = resnet_model.fit(train_generator,
   batch_size=batch_size,
66.                   validation_data=validation_generator,
67.                   steps_per_epoch=train_steps,
68.                   validation_steps=None,
69.                   epochs=epoch,
70.                   callbacks=callback_list
71.   )
```

Line 44 – 60 is used to make the model for this code that shown is for ResNet50. For this experiment, the author is using ResNet50V2 because it has better accuracy than ResNet50. If the code was used for EfficientNet the code change on line 46 from tf.keras.applications.resnet_v2.ResNet50V2 into tf.keras.applications.EfficientNetB0 if using EfficientNetB0, tf.keras.applications.EfficientNetB1 if using EfficientNetB1, and others until EfficientNetB7. In line 46 tf.keras.applications.resnet_v2.ResNet50V2 was the model that the author compared to the other. The weight in line 46 must be on imagenet because in this experiment was comparing between pre-trained models. But when the author is using ResNet50V2 on the Kaggle Notebook there are some problems that the weight imagenet on ResNet50V2 cannot be loaded automatically so the author downloads the h5 that is provided and uploads it to Kaggle then load it manually. Because the output from model like ResNet50 cannot have the output of exact number of classes, the author using additional layer so it can have the exact output that same to the number of classes. In this experiment Dropout layer are used to reducing the overfitting. Dropout layer are mask that ignoring some neurons towards next layers. Dropout sets input units to 0 randomly with frequency that has been input on each step during training. For the example Dropout(0.2) mean layer will be set input unit to with frequency 0.2 at each step. With using Dropout layer it help to reducing or prevented overfitting. Dropout are only applies when training. In this experiment, the author also compared additional models with different additional layers. For the first additional model, the author uses Flatten and Drop 0.2 as an additional layer. For the next additional model, the author uses Flatten and Drop 0.5 as an additional layer. After that the author uses GlobalAveragePooling2D and Drop 0.2 as an additional layer. Next one the author uses GlobalAveragePooling2D and Drop 0.5 as an additional layer. The author also uses GlobalAveragePooling2D, BatchNormalization and Drop 0.5 as an additional layer for the next additional model. Next, the author uses GlobalAveragePooling2D, BatchNormalization, Drop 0.5, Dense (256), and Drop 0.5 as an additional layer for the next additional model. And lastly, the author uses GlobalAveragePooling2D, BatchNormalization, Drop 0.5, Dense (512), and Drop 0.5 as an additional layer for the next additional model. In line 62-64 the author uses reduce learning rate if the validation loss did not improve and uses checkpoint to save the best training that has been achieved.

4.3. Results

All the results that have been acquired are using almost the same parameter for each model. The only difference is only happening to EfficientNet B7 because when the author tries to train the model there are some issues. To fix the issue the author reduced the batch size from 32 to 16 only for the EfficientNet B7. The result from the first additional model the author uses Flatten and Drop 0.2 as an additional layer.

Table 4.2. Accuracy Table Flatten – Drop 0.2

Model	Train	Validation	Test
ResNet50	94.56%	78.56%	80.72%
EfficientNet B0	98.73%	86.43%	84.12%
EfficientNet B1	98.36%	85.73%	86.41%
EfficientNet B2	99.04%	87.92%	87.01%
EfficientNet B3	97.64%	85.53%	84.62%
EfficientNet B4	98.98%	84.93%	85.01%
EfficientNet B5	98.03%	85.93%	84.52%
EfficientNet B6	99.45%	85.93%	85.51%
EfficientNet B7	98.94%	85.93%	86.91%

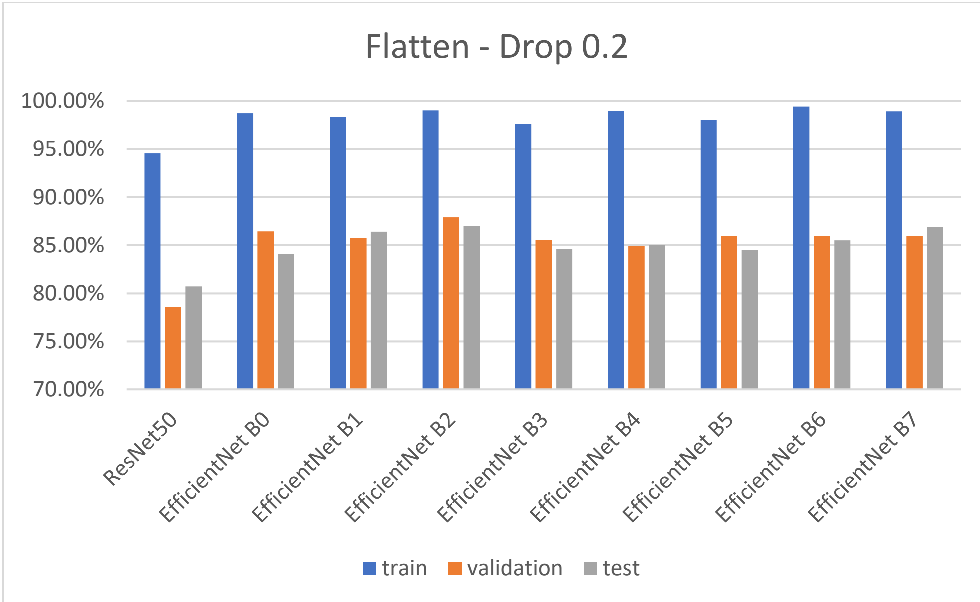


Figure 4.1 Graph Flatten – Drop 0.2

From the first additional model, which is Flatten – Drop 0.2 the Table 4.2 and figure 4.1, it reveals that the training accuracy is a lot higher than the validation and test accuracy. The lowest validation and test accuracy acquired by ResNet50 achieved 94.56% on train accuracy, 78.56% on validation accuracy and 80.72% on test accuracy. The highest validation accuracy, and test accuracy acquired by EfficientNet B2 achieved 99.04% on train accuracy, 87.92% on validation accuracy, and 87.01% on test accuracy.

Table 4.3. Accuracy Table Flatten – Drop 0.5

Model	Train	Validation	Test
ResNet50	97.54%	82.24%	80.54%
EfficientNet B0	98.90%	86.03%	84.62%
EfficientNet B1	98.69%	85.93%	85.21%
EfficientNet B2	98.83%	85.73%	86.71%
EfficientNet B3	99.13%	87.33%	86.31%
EfficientNet B4	98.68%	86.33%	87.01%
EfficientNet B5	98.75%	83.33%	84.72%
EfficientNet B6	99.06%	84.13%	84.92%
EfficientNet B7	99.08%	86.33%	86.11%

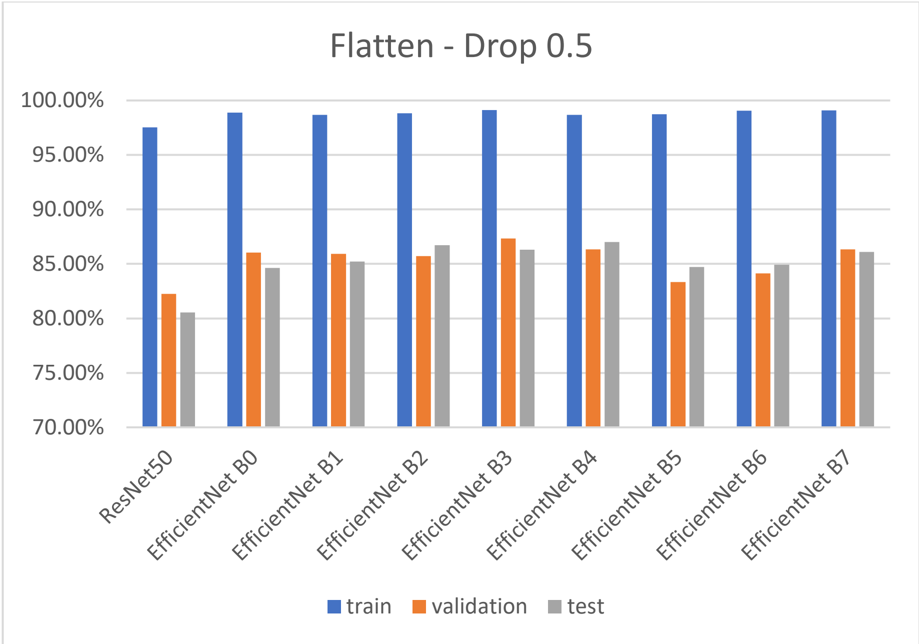


Figure 4.2 Graph Flatten – Drop 0.5

The second additional layers model is Flatten – Drop 0.5 on Table 4.3 and figure 4.2 can be seen that training accuracy is way higher than validation accuracy and test accuracy. The lowest validation accuracy and test accuracy was achieved by ResNet 50 which got 97.54% on train accuracy, 82.24% on validation accuracy and 80.54% on test accuracy. The highest test accuracy was achieved by EfficientNet B4 which got 98.68% on train accuracy. 86.33% on validation accuracy, and 87.01% on test accuracy.

Table 4.4. Accuracy Table GlobalAveragePooling2D – Drop 0.2

Model	Train	Validation	Test
ResNet50	97.90%	82.44%	79.52%
EfficientNet B0	98.34%	86.73%	85.51%
EfficientNet B1	98.60%	85.63%	85.91%
EfficientNet B2	98.61%	88.02%	87.91%
EfficientNet B3	99.13%	88.52%	86.41%
EfficientNet B4	99.20%	86.93%	85.51%
EfficientNet B5	98.88%	83.33%	85.51%
EfficientNet B6	99.20%	85.43%	88.21%
EfficientNet B7	98.83%	86.43%	87.81%

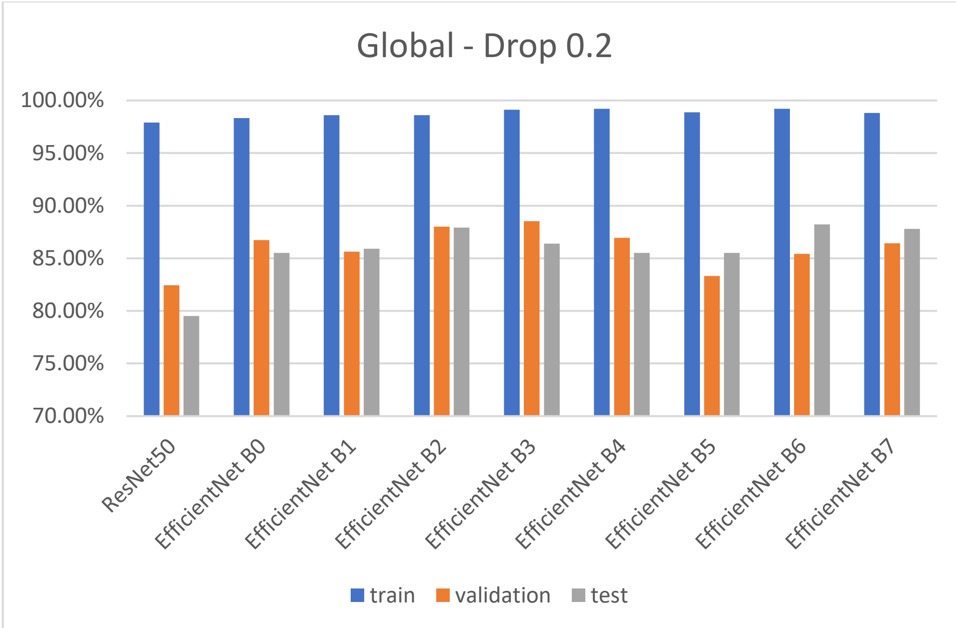


Figure 4.3 Graph GlobalAveragePooling2D – Drop 0.2

The next additional layers model is GlobalAveragePooling2D – Drop 0.2 on the Table 4.4 and figure 4.3 showing that training accuracy is way higher than validation accuracy and test accuracy. The lowest validation accuracy and test accuracy was achieved by ResNet50 which got 97.90% on train accuracy, 82.44% on validation accuracy, and 79.52% on test accuracy. The highest test accuracy was achieved by EfficientNet B6 which got 99.20% on train accuracy. 85.43% on validation accuracy, and 88.21% on test accuracy.

Table 4.5. Accuracy Table GlobalAveragePooling2D – Drop 0.5

Model	Train	Validation	Test
ResNet50	90.66%	83.33%	81.52%
EfficientNet B0	98.19%	86.43%	84.42%
EfficientNet B1	98.65%	87.43%	87.81%
EfficientNet B2	96.83%	86.73%	87.91%
EfficientNet B3	98.90%	85.43%	86.91%
EfficientNet B4	99.38%	87.33%	87.01%
EfficientNet B5	98.86%	86.03%	84.32%
EfficientNet B6	99.14%	85.33%	84.72%
EfficientNet B7	98.90%	86.13%	85.21%

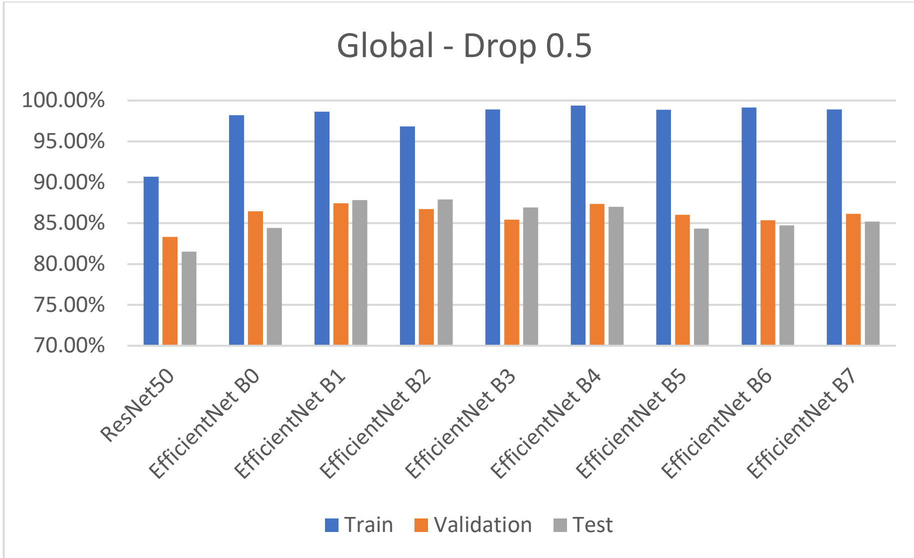


Figure 4.4 Graph GlobalAveragePooling2D – Drop 0.5

On Table 4.5 and Figure 4.4 with additional layer model GlobalAveragePooling2D – Drop 0.5 the highest validation accuracy acquired by EfficientNet B1 acquire 98.65% on train accuracy, 87.43% on validation accuracy, and 87.81% on test accuracy. However the highest test accuracy was acquired by EfficientNet B2 with 0.1% difference accuracy that acquired 96.83% on train accuracy, 86.73% on validation accuracy, and 87.91% on test accuracy. The worst validation accuracy and test accuracy was acquired by ResNet50 that have 90.66% on train accuracy, 83.33% on validation accuracy, and 81.52% on test accuracy.

Table 4.6. Accuracy Table GlobalAveragePooling2D – BatchNormalization – Drop 0.5

Model	Train	Validation	Test
ResNet50	96.83%	81.34%	78.62%
EfficientNet B0	98.56%	85.93%	83.32%
EfficientNet B1	98.73%	85.63%	85.01%
EfficientNet B2	98.75%	87.92%	84.92%
EfficientNet B3	98.44%	86.93%	87.51%
EfficientNet B4	99.19%	85.43%	86.11%
EfficientNet B5	98.86%	85.43%	84.02%
EfficientNet B6	98.54%	85.23%	84.52%
EfficientNet B7	98.74%	86.33%	84.72%

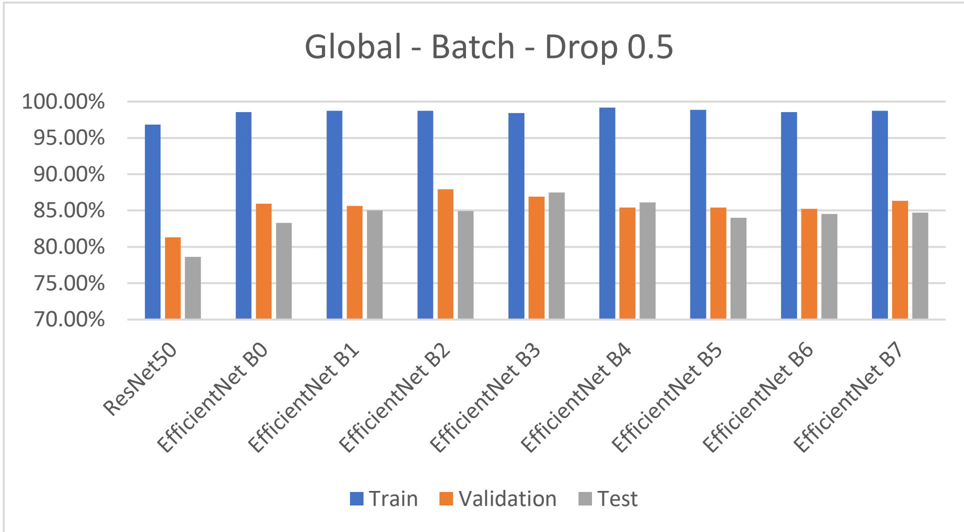


Figure 4.5 Graph GlobalAveragePooling2D – BatchNormalization – Drop 0.5

The fifth additional layers model is GlobalAveragePooling2D –- BatchNormalization - Drop 0.5 on the Table 4.6 and figure 4.5 can be seen that training accuracy is way higher than validation accuracy and test accuracy. The lowest validation accuracy and test accuracy was achieved by ResNet50 which got 96.83% on train accuracy, 81.34% on validation accuracy, and 78.62% on test accuracy. The highest validation accuracy acquired by EfficientNet B2 was 98.75% on train accuracy, 87.92% on validation accuracy, and 84.92% on test accuracy. But the highest test accuracy was acquired by EfficientNet B3 with a 2.59 % difference accuracy with EfficientNet B2 which acquired 98.44% in train accuracy, 86.93% in validation accuracy, and 87.51% in test accuracy.

Table 4.7. Accuracy Table Global – Batch – Drop 0.5 – Dense 256 – Drop 0.5

Model	Train	Validation	Test
ResNet50	97.79%	83.83%	83.42%
EfficientNet B0	97.68%	85.63%	84.82%
EfficientNet B1	98.45%	85.03%	85.21%
EfficientNet B2	98.13%	88.02%	84.12%
EfficientNet B3	98.68%	85.83%	85.61%
EfficientNet B4	98.66%	85.53%	86.71%
EfficientNet B5	98.70%	84.63%	84.72%
EfficientNet B6	98.79%	84.03%	85.81%
EfficientNet B7	97.14%	86.23%	88.41%

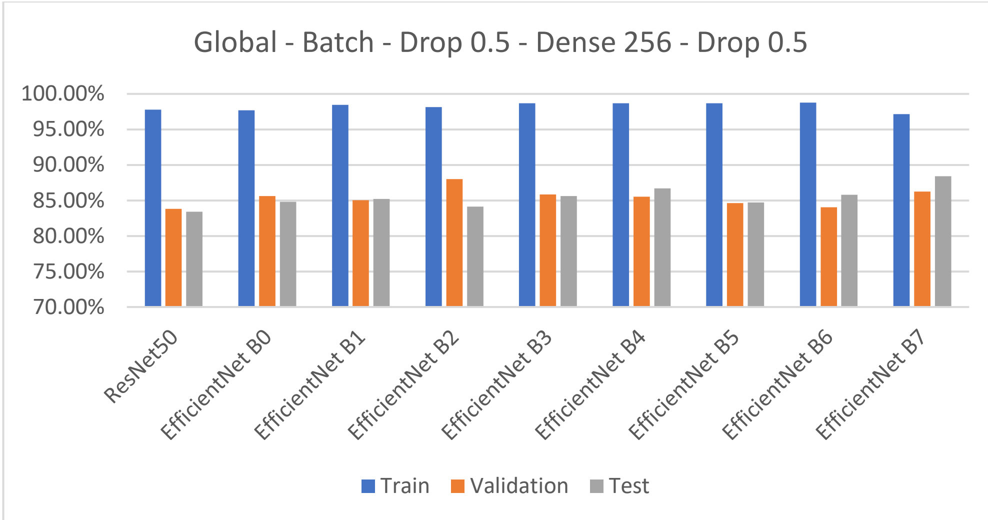


Figure 4.6 Graph Global – Batch – Drop 0.5 – Dense 256 – Drop 0.5

The next additional layers model is GlobalAveragePooling2D –- BatchNormalization - Drop 0.5 – Dense 256 – Drop 0.5 on the Table 4.6 and figure 4.5 showing that training accuracy is way higher than validation accuracy and test accuracy. The lowest validation accuracy and test accuracy was achieved by ResNet50 which got 97.79% on train accuracy, 83.83% on validation accuracy, and 83.42% on test accuracy. The highest validation accuracy acquired by EfficientNet B2 was 98.13% on train accuracy, 88.02% on validation accuracy, and 84.12% on test accuracy. But the highest test accuracy was acquired by EfficientNet B7 with 4.29 % difference accuracy with EfficientNet B2 which acquired 97.14% on train accuracy, 86.23% on validation accuracy, and 88.41%on test accuracy.

Table 4.8. Accuracy Table Global – Batch – Drop 0.5 – Dense 512 – Drop 0.5

Model	Train	Validation	Test
ResNet50	96.69%	79.54%	78.82%
EfficientNet B0	97.77%	86.13%	86.11%
EfficientNet B1	98.02%	86.03%	85.01%
EfficientNet B2	98.29%	86.13%	85.31%
EfficientNet B3	98.54%	86.63%	85.91%
EfficientNet B4	98.85%	84.03%	86.41%
EfficientNet B5	98.84%	85.23%	83.32%
EfficientNet B6	98.79%	84.23%	85.31%
EfficientNet B7	97.70%	84.43%	85.81%

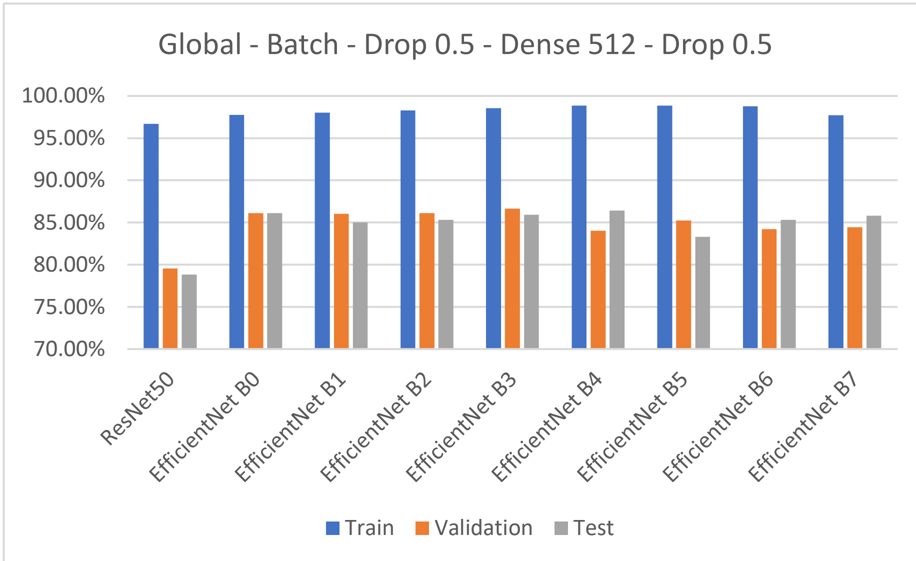


Figure 4.7 Graph Global – Batch – Drop 0.5 – Dense 512 – Drop 0.5

The last additional layers model is GlobalAveragePooling2D –- BatchNormalization - Drop 0.5 – Dense 512 – Drop 0.5 on the Table 4.6 and figure 4.5 showing the highest validation accuracy acquired by EfficientNet B3 acquired 98.54% on train accuracy, 86.63% on validation accuracy, and 85.91% on test accuracy. But the highest test accuracy was acquired by EfficientNet B4 with a 0.5% difference accuracy with EfficientNet B3 which acquired 98.85% in train accuracy, 84.03% in validation accuracy, and 86.41% in test accuracy. The lowest validation accuracy and test accuracy was achieved by ResNet50 which got 96.69% on train accuracy, 79.54% on validation accuracy, and 78.82% on test accuracy.

4.4. Discussion

From the accuracy obtained from before that listed on from table 4.2 to table 4.8 it is showing that the training accuracy was so much higher than the validation accuracy and test accuracy. It also shows that the validation accuracy and test accuracy never surpass 90% accuracy. It might happen because the dataset that has a different amount of data for each class like the class melanocytic nevi has 6705 images of it but class dermatofibroma only has 115 images of it. What make it worse is because the number of that before splitting the data into 80% training, 10% validation, and 10% test.

Table 4.9. Best Test Accuracy

Model	Test	Additional Layer
ResNet50	83.42%	Global - Batch - Drop 0.5 - Dense 256 - Drop 0.5
EfficientNet B0	86.11%	Global - Batch - Drop 0.5 - Dense 512 - Drop 0.5
EfficientNet B1	87.81%	Global - Batch - Drop 0.5
EfficientNet B2	87.91%	Global - Drop 0.2
EfficientNet B3	87.51%	Global - Batch - Drop 0.5
EfficientNet B4	87.01%	Global - Drop 0.5
EfficientNet B5	85.51%	Global - Drop 0.2
EfficientNet B6	88.21%	Global - Drop 0.2
EfficientNet B7	88.41%	Global - Batch - Drop 0.5 - Dense 256 - Drop 0.5

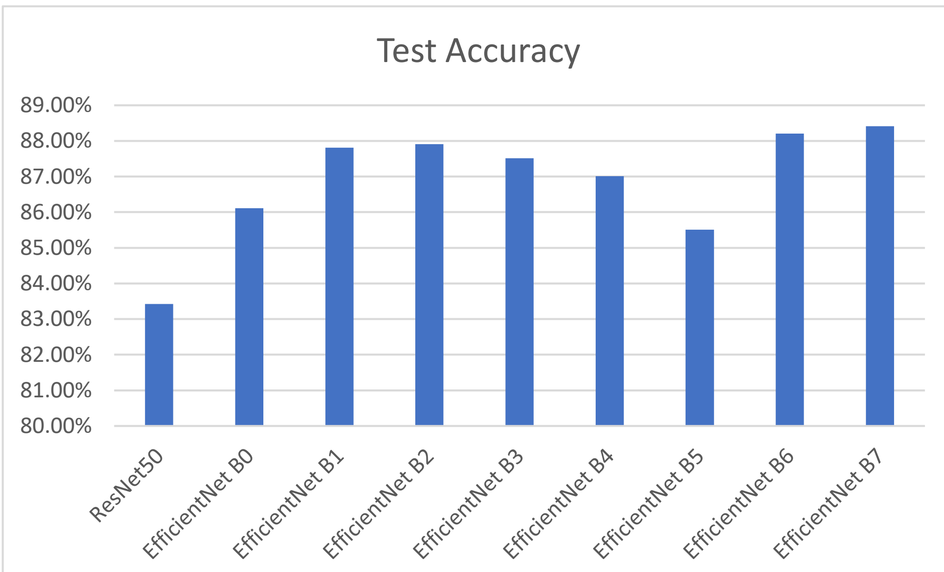


Figure 4.8 Graph Best Test Accuracy

These are the best results of the test accuracy shown in table 4.9 and figure 4.8. It shows that the lowest test accuracy is from ResNet50 with 83.42% accuracy and the highest accuracy obtained from EfficientNet B7 with 88.41% accuracy. it shows that the ResNet 50 is not very good at classifying skin cancer. It also shows that when using additional layers like Flatten - Drop 0.5 or Flatten – Drop 0.2 the accuracy obtained cannot be maximized.

.

CHAPTER 5
CONCLUSION

In this research, the author compares between ResNet50 and EfficientNet B0-B7 on the classification of skin cancer. From the result, it can be concluded that both ResNet and EfficientNet can be used to classify skin cancer. but on the test and validation EfficientNet Outperforms ResNet 50. When on the best additional layer model that this experiment does the EfficientNet got the highest score on test accuracy that was acquired by EfficientNet B7 on 88.41% accuracy. But when the ResNet50 on the best additional layer model, this experiment only acquired 83.42% accuracy. From that, the author can conclude that EfficientNet was better at the classification of skin cancer.

This research can be enhanced more by changing the dataset or adding the dataset because this dataset that used for the experiment is very unbalanced like the class melanocytic nevi have 6705 images of it but class dermatofibroma only has 115 images of it. Using other pre-trained algorithms can also be used to compare the accuracy of the EfficientNet like DenseNet and many others. It can be also compared between using pre-processing or not or using other pre-processing methods.

REFERENCES

[1] H. Huang, B. W. Hsu, C. Lee, and V. S. Tseng, 'Development of a light-weight deep learning model for cloud applications and remote diagnosis of skin cancers', *J. Dermatol.*, vol. 48, no. 3, pp. 310–316, Mar. 2021, doi: 10.1111/1346-8138.15683.

[2] K. Ali, Z. A. Shaikh, A. A. Khan, and A. A. Laghari, 'Multiclass skin cancer classification using EfficientNets – a first step towards preventing skin cancer', *Neuroscience Informatics*, vol. 2, no. 4, p. 100034, Dec. 2022, doi: 10.1016/j.neuri.2021.100034.

[3] Md. K. Islam *et al.*, 'Melanoma Skin Lesions Classification using Deep Convolutional Neural Network with Transfer Learning', in *2021 1st International Conference on Artificial Intelligence and Data Analytics (CAIDA)*, Riyadh, Saudi Arabia: IEEE, Apr. 2021, pp. 48–53. doi: 10.1109/CAIDA51941.2021.9425117.

[4] M. Tahir, A. Naeem, H. Malik, J. Tanveer, R. A. Naqvi, and S.-W. Lee, 'DSCC_Net: Multi-Classification Deep Learning Models for Diagnosing of Skin Cancer Using Dermoscopic Images', *Cancers*, vol. 15, no. 7, p. 2179, Apr. 2023, doi: 10.3390/cancers15072179.

[5] M. S. Ali, M. S. Miah, J. Haque, M. M. Rahman, and M. K. Islam, 'An enhanced technique of skin cancer classification using deep convolutional neural network with transfer learning models', *Machine Learning with Applications*, vol. 5, p. 100036, Sep. 2021, doi: 10.1016/j.mlwa.2021.100036.

[6] D. Popescu, M. El-khatib, and L. Ichim, 'Skin Lesion Classification Using Collective Intelligence of Multiple Neural Networks', *Sensors*, vol. 22, no. 12, Art. no. 12, Jan. 2022, doi: 10.3390/s22124399.

[7] A. Khamparia, P. K. Singh, P. Rani, D. Samanta, A. Khanna, and B. Bhushan, 'An internet of health things-driven deep learning framework for detection and classification of skin cancer using transfer learning', *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 7, p. e3963, 2021, doi: 10.1002/ett.3963.

[8] S. Mohapatra, N. v. s. Abhishek, D. Bardhan, A. A. Ghosh, and S. Mohanty, 'Comparison of MobileNet and ResNet CNN Architectures in the CNN-Based Skin Cancer Classifier Model', in *Machine Learning for Healthcare Applications*, John Wiley & Sons, Ltd, 2021, pp. 169–186. doi: 10.1002/9781119792611.ch11.

[9] N. Abuared, A. Panthakkan, M. Al-Saad, S. A. Amin, and W. Mansoor, 'Skin Cancer Classification Model Based on VGG 19 and Transfer Learning', in *2020 3rd International Conference on Signal Processing and Information Security (ICSPIS)*, DUBAI, United Arab Emirates: IEEE, Nov. 2020, pp. 1–4. doi: 10.1109/ICSPIS51252.2020.9340143.

[10] A. Tajerian, M. Kazemian, M. Tajerian, and A. A. Malayeri, 'Design and validation of a new machine-learning-based diagnostic tool for the differentiation of dermatoscopic skin cancer images', *PLOS ONE*, vol. 18, no. 4, p. e0284437, Apr. 2023, doi: 10.1371/journal.pone.0284437.

[11] A. Rastogi, 'ResNet50', Medium. Accessed: Jun. 21, 2023. [Online]. Available: https://blog.devgenius.io/resnet50-6b42934db431

APPENDIX

IMPORT LIBRARIES

```
import os
import cv2
import shutil
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.layers import Input, BatchNormalization, GlobalAveragePooling2D, Dense, Dropout,Flatten
from keras.models import Model, load_model
from keras.applications.resnet_v2 import ResNet50V2
import tensorflow as tf
from tensorflow.keras import regularizers
```

MEMBUAT FUNGSI UNTUK MENAMPILKAN GRAFIK

```
def plot_history(history, title, plot_type, xlabel, ylabel):
    plt.figure(figsize=(12,8))
    if plot_type == 'accuracy':
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.legend(['Train Accuracy', 'Validation Accuracy'], loc='upper left')
    else:
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.legend(['Train Loss', 'Validation Loss'], loc='upper left')
    plt.title('Model Accuracy')
    plt.ylabel('{}'.format(ylabel))
    plt.xlabel('{}'.format(xlabel))
    plt.gca().ticklabel_format(axis='both', style='plain', useOffset=False)
    plt.title('{}'.format(title))
    plt.savefig('{}.png'.format(title))
    plt.show()
```

MEMISAHKAN DATA KE SETIAP CLASS DAN PREPROCESSING

```
df = pd.read_csv('/kaggle/input/ham1000-segmentation-and-classification/GroundTruth.csv')
classes=df.columns[1:]
parent_folder="baru"
if os.path.isdir(parent_folder):
    shutil.rmtree(parent_folder)
os.mkdir(parent_folder)
for i in classes:
    folder=os.path.join(parent_folder,i)
    os.mkdir(folder)
```

```
def preprocessing(image):
    gambar=cv2.imread(f"/kaggle/input/ham1000-segmentation-and-classification/images/{image}.jpg",cv2.IMREAD_COLOR)
    resize=cv2.resize(gambar,[224,224])
    grayScale = cv2.cvtColor(resize, cv2.COLOR_RGB2GRAY )
    #Black hat filter
    kernel = cv2.getStructuringElement(1,(9,9))
    blackhat = cv2.morphologyEx(grayScale, cv2.MORPH_BLACKHAT, kernel)
    #Gaussian blur
    bhg= cv2.GaussianBlur(blackhat,(3,3),cv2.BORDER_DEFAULT)
    #masking
    ret,mask = cv2.threshold(bhg,10,255,cv2.THRESH_BINARY)
    #Replace pixels of the mask
    dst = cv2.inpaint(resize,mask,6,cv2.INPAINT_TELEA)
    return dst
for cls in classes:
    images = df[df[cls]==1]['image'].to_list()
    for image in images:
        gambar=preprocessing(image)
        cv2.imwrite(f"baru/{cls}/{image}.jpg",gambar)
```

SPLITTING DATA MENJADI TRAIN, TEST, VALIDATION

```
def splitting(location):
    files=[]
    labels=[]
    classes=os.listdir(location)
    for cls in classes:
        path=os.path.join(location,cls)
        list_file=os.listdir(path)
        for file in list_file:
            file_path=os.path.join(path,file)
            files.append(file_path)
            labels.append(cls)
    filepath=pd.Series(files,name="path")
    labelpath=pd.Series(labels,name="label")
    dataset=pd.concat([filepath,labelpath],axis=1)
    strat=dataset['label']
    train_dataset,test_valid_dataset=train_test_split(dataset,train_size=0.8,shuffle=True,random_state=42,stratify=strat)
    strat_test_valid=test_valid_dataset["label"]
    test_dataset,validation_dataset=train_test_split(test_valid_dataset,train_size=0.5,shuffle=True,random_state=42,stratify=strat_test_valid)
    print('train_df length: ', len(train_dataset), '  test_df length: ',len(test_dataset), '  valid_df length: ', len(validation_dataset))
    print(train_dataset['label'].value_counts())
    return train_dataset,test_dataset,validation_dataset
train_dataset,test_dataset,validation_dataset=splitting(parent_folder)
```

MEMBUAT DATA GENERATOR UNTUK TRAIN TEST VALIDATION

```
batch_size=32
img_size=(224,224)
```

```
    img_shape=(224,224,3)
    length=len(test_dataset)
    learning_rate=1e-4
    epoch=30
    factor=learning_rate/epoch
    test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length % n ==0 and length/n<=64],reverse=True)[0]
    test_steps=int(length/test_batch_size)
    print ( 'test batch size: ' ,test_batch_size, '  test steps: ', test_steps)
    def process(image):
        return image
    generator_for_training=ImageDataGenerator(preprocessing_function=process,horizontal_flip=True)
    generator=ImageDataGenerator(preprocessing_function=process)

    train_generator=generator_for_training.flow_from_dataframe(train_dataset,x_col='path',y_col='label',
target_size=img_size,class_mode='categorical',color_mode='rgb',
shuffle=True,batch_size=batch_size)
    validation_generator=generator.flow_from_dataframe(validation_dataset,x_col='path',y_col='label',
target_size=img_size,class_mode='categorical',color_mode='rgb',
shuffle=True,batch_size=batch_size)
    test_generator=generator.flow_from_dataframe(test_dataset,x_col='path',y_col='label',
target_size=img_size,class_mode='categorical',color_mode='rgb',
shuffle=False,batch_size=test_batch_size)
    train_steps=int(np.ceil(len(train_generator.labels)/batch_size))
```

MEMBUAT MODEL ResNet-50

```
    def resnet_model1():
        input=Input(shape=img_shape)
        base=ResNet50V2(weights='/kaggle/input/resnetv2-weight/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5',include_top=False,input_shape=img_shape,input_tensor=input,classes=7)

        base.trainable=False
    #     x=Flatten()(base.output)
        x = GlobalAveragePooling2D()(base.output)
        x = BatchNormalization()(x)
    #
    #     x = Dropout(0.2)(x)
        x = Dropout(0.5)(x)
    #     x = Dense(512, activation='relu')(x)
    #     x = Dropout(0.5)(x)
    #     x = Dense(256, activation='relu')(x)
    #     x = Dropout(0.5)(x)
```

```
    output = Dense(7, activation='softmax',kernel_regularizer=regularizers.L1L2(l1=0.01, l2=0.01))(x)
    model = Model(input, output)
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss="categorical_crossentropy",  metrics=['accuracy'])
    return model
resnet_model=resnet_model1()
```

TRAINING MODEL ResNet-50

```
reduce_lr   =tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=factor,patience=5, min_lr=learning_rate,verbose=1)
checkpoint=tf.keras.callbacks.ModelCheckpoint("resnet_model.h5",monitor="val_loss",mode="min",save_best_only = True,verbose=1)
callback_list = [ reduce_lr, checkpoint]
classifier_history = resnet_model.fit(train_generator, batch_size=batch_size,
                    validation_data=validation_generator,
                    steps_per_epoch=train_steps,
                    validation_steps=None,
                    epochs=epoch,
                    callbacks=callback_list
)
resnet_model.save("resnet_model_coba.h5")

```

MEMBUAT GRAFIK ResNet-50, MENGETAHUI AKURASI UNTUK TEST DATASET

```
plot_history(classifier_history, 'Accuracy', 'accuracy', 'Epochs', 'Model Accuracy')
plot_history(classifier_history, 'Loss', 'loss', 'Epochs', 'Loss')
acc=resnet_model.evaluate( test_generator, batch_size=test_batch_size, verbose=1, steps=test_steps, return_dict=False)[1]*100
print(f'accuracy on the test set is {acc:5.2f} %')
```

MEMBUAT MODEL EfficientNet-B0

```
def efficientnet_b0():
    input=Input(shape=img_shape)
base=tf.keras.applications.EfficientNetB0(weights='imagenet',include_top=False,input_shape=img_shape,input_tensor=input,classes=7)
#     base.trainable=False
#     x=Flatten()(base.output)
    x = GlobalAveragePooling2D()(base.output)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
#     x = Dense(256, activation='relu')(x)
#     x = Dropout(0.5)(x)
```

```
    output = Dense(7, activation='softmax',kernel_regularizer=regularizers.L1L2(l1=0.01, l2=0.01))(x)
    model = Model(input, output)
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss="categorical_crossentropy",  metrics=['accuracy'])
    return model
efficientnet_b0_model=efficientnet_b0()
```

TRAINING MODEL EfficientNet-B0

```
reduce_lr =tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=factor,patience=5, min_lr=learning_rate,verbose=1)
checkpoint=tf.keras.callbacks.ModelCheckpoint("efficientnet_b0_model.h5",monitor="val_loss",mode="min",save_best_only = True,verbose=1)
callback_list = [ reduce_lr, checkpoint]
classifier_history = efficientnet_b0_model.fit(train_generator, batch_size=batch_size,
                validation_data=validation_generator,
                steps_per_epoch=train_steps,
                validation_steps=None,
                epochs=epoch,
                callbacks=callback_list
)
efficientnet_b0_model.save("efficientnet_b0_model_coba.h5")

```

MEMBUAT GRAFIK EfficientNet-B0, MENGETAHUI AKURASI UNTUK TEST DATASET

```
plot_history(classifier_history, 'Accuracy', 'accuracy', 'Epochs', 'Model Accuracy')
plot_history(classifier_history, 'Loss', 'loss', 'Epochs', 'Loss')
acc=efficientnet_b0_model.evaluate( test_generator, batch_size=test_batch_size, verbose=1, steps=test_steps, return_dict=False)[1]*100
print(f'accuracy on the test set is {acc:5.2f} %')
```

MEMBUAT MODEL EfficientNet-B1

```
def efficientnet_b1():
    input=Input(shape=img_shape)
base=tf.keras.applications.EfficientNetB1(weights='imagenet',include_top=False,input_shape=img_shape,input_tensor=input,classes=7)
#     base.trainable=False
#     x=Flatten()(base.output)
    x = GlobalAveragePooling2D()(base.output)
    x = BatchNormalization()(x)
#     x = Dropout(0.2)(x)
    x = Dropout(0.5)(x)
#     x = Dense(512, activation='relu')(x)
#     x = Dropout(0.5)(x)
#     x = Dense(256, activation='relu')(x)
#     x = Dropout(0.5)(x)
```

```

    output = Dense(7, activation='softmax',kernel_regularizer=regularizers.L1L2(l1=0.01, l2=0.01))(x)
    model = Model(input, output)
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss="categorical_crossentropy",  metrics=['accuracy'])
    return model
efficientnet_b1_model=efficientnet_b1()
```

TRAINING MODEL EfficientNet-B1

```
reduce_lr  =tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=factor,patience=5, min_lr=learning_rate,verbose=1)
checkpoint=tf.keras.callbacks.ModelCheckpoint("efficientnet_b1_model.h5",monitor="val_loss",mode="min",save_best_only = True,verbose=1)
callback_list = [ reduce_lr, checkpoint]
classifier_history = efficientnet_b1_model.fit(train_generator, batch_size=batch_size,
                    validation_data=validation_generator,
                    steps_per_epoch=train_steps,
                    validation_steps=None,
                    epochs=epoch,
                    callbacks=callback_list
)
efficientnet_b1_model.save("efficientnet_b1_model_coba.h5")
```

MEMBUAT GRAFIK EfficientNet-B1, MENGETAHUI AKURASI UNTUK TEST DATASET

```
plot_history(classifier_history, 'Accuracy', 'accuracy', 'Epochs', 'Model Accuracy')
plot_history(classifier_history, 'Loss', 'loss', 'Epochs', 'Loss')
acc=efficientnet_b1_model.evaluate( test_generator, batch_size=test_batch_size, verbose=1, steps=test_steps, return_dict=False)[1]*100
print(f'accuracy on the test set is {acc:5.2f} %')
```

MEMBUAT MODEL EfficientNet-B2

```
def efficientnet_b2():
    input=Input(shape=img_shape)
    base=tf.keras.applications.EfficientNetB2(weights='imagenet',include_top=False,input_shape=img_shape,input_tensor=input,classes=7)
#     base.trainable=False
#     x=Flatten()(base.output)
    x = GlobalAveragePooling2D()(base.output)
    x = BatchNormalization()(x)
#     x = Dropout(0.2)(x)
    x = Dropout(0.5)(x)
#     x = Dense(512, activation='relu')(x)
#     x = Dropout(0.5)(x)
```

```
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(7, activation='softmax',kernel_regularizer=regularizers.L1L2(l1=0.01, l2=0.01))(x)
    model = Model(input, output)
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=['accuracy'])
    return model
efficientnet_b2_model=efficientnet_b2()
```

TRAINING MODEL EfficientNet-B2

```
reduce_lr =tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=factor,patience=5, min_lr=learning_rate,verbose=1)
checkpoint=tf.keras.callbacks.ModelCheckpoint("efficientnet_b2_model.h5",monitor="val_loss",mode="min",save_best_only = True,verbose=1)
callback_list = [ reduce_lr, checkpoint]
classifier_history = efficientnet_b2_model.fit(train_generator, batch_size=batch_size,
                validation_data=validation_generator,
                steps_per_epoch=train_steps,
                validation_steps=None,
                epochs=epoch,
                callbacks=callback_list
)
efficientnet_b2_model.save("efficientnet_b2_model_coba.h5")
```

MEMBUAT GRAFIK EfficientNet-B2, MENGETAHUI AKURASI UNTUK TEST DATASET

```
plot_history(classifier_history, 'Accuracy', 'accuracy', 'Epochs', 'Model Accuracy')
plot_history(classifier_history, 'Loss', 'loss', 'Epochs', 'Loss')
acc=efficientnet_b2_model.evaluate( test_generator, batch_size=test_batch_size, verbose=1, steps=test_steps, return_dict=False)[1]*100
print(f'accuracy on the test set is {acc:5.2f} %')
```

MEMBUAT MODEL EfficientNet-B3

```
def efficientnet_b3():
    input=Input(shape=img_shape)
    base=tf.keras.applications.EfficientNetB3(weights='imagenet',include_top=False,input_shape=img_shape,input_tensor=input,classes=7)
#     base.trainable=False
#     x=Flatten()(base.output)
    x = GlobalAveragePooling2D()(base.output)
#     x = BatchNormalization()(x)
#     x = Dropout(0.2)(x)
    x = Dropout(0.5)(x)
```

```
#     x = Dense(512, activation='relu')(x)
#     x = Dropout(0.5)(x)
#     x = Dense(256, activation='relu')(x)
#     x = Dropout(0.5)(x)

    output = Dense(7, activation='softmax',kernel_regularizer=regularizers.L1L2(l1=0.01, l2=0.01))(x)
    model = Model(input, output)
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss="categorical_crossentropy",  metrics=['accuracy'])
    return model
efficientnet_b3_model=efficientnet_b3()
```

TRAINING MODEL EfficientNet-B3

```
reduce_lr  =tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=factor,patience=5, min_lr=learning_rate,verbose=1)
checkpoint=tf.keras.callbacks.ModelCheckpoint("efficientnet_b3_model.h5",monitor="val_loss",mode="min",save_best_only = True,verbose=1)
callback_list = [ reduce_lr, checkpoint]
classifier_history = efficientnet_b3_model.fit(train_generator, batch_size=batch_size,
                  validation_data=validation_generator,
                  steps_per_epoch=train_steps,
                  validation_steps=None,
                  epochs=epoch,
                  callbacks=callback_list
)
efficientnet_b3_model.save("efficientnet_b3_model_coba.h5")
```

MEMBUAT GRAFIK EfficientNet-B3, MENGETAHUI AKURASI UNTUK TEST DATASET

```
plot_history(classifier_history, 'Accuracy', 'accuracy', 'Epochs', 'Model Accuracy')
plot_history(classifier_history, 'Loss', 'loss', 'Epochs', 'Loss')
acc=efficientnet_b3_model.evaluate( test_generator, batch_size=test_batch_size, verbose=1, steps=test_steps, return_dict=False)[1]*100
print(f'accuracy on the test set is {acc:5.2f} %')
```

MEMBUAT MODEL EfficientNet-B4

```
def efficientnet_b4():
    input=Input(shape=img_shape)
base=tf.keras.applications.EfficientNetB4(weights='imagenet',include_top=False,input_shape=img_shape,input_tensor=input,classes=7)
#     base.trainable=False
#     x=Flatten()(base.output)
    x = GlobalAveragePooling2D()(base.output)
    x = BatchNormalization()(x)
#     x = Dropout(0.2)(x)
    x = Dropout(0.5)(x)
```

```
#     x = Dense(512, activation='relu')(x)
#     x = Dropout(0.5)(x)
#     x = Dense(256, activation='relu')(x)
#     x = Dropout(0.5)(x)

    output = Dense(7, activation='softmax',kernel_regularizer=regularizers.L1L2(l1=0.01, l2=0.01))(x)
    model = Model(input, output)
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss="categorical_crossentropy",  metrics=['accuracy'])
    return model
efficientnet_b4_model=efficientnet_b4()
```

TRAINING MODEL EfficientNet-B4

```
reduce_lr =tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=factor,patience=5, min_lr=learning_rate,verbose=1)
checkpoint=tf.keras.callbacks.ModelCheckpoint("efficientnet_b4_model.h5",monitor="val_loss",mode="min",save_best_only = True,verbose=1)
callback_list = [ reduce_lr, checkpoint]
classifier_history = efficientnet_b4_model.fit(train_generator, batch_size=batch_size,
                  validation_data=validation_generator,
                  steps_per_epoch=train_steps,
                  validation_steps=None,
                  epochs=epoch,
                  callbacks=callback_list
)
efficientnet_b4_model.save("efficientnet_b4_model_coba.h5")
```

MEMBUAT GRAFIK EfficientNet-B4, MENGETAHUI AKURASI UNTUK TEST DATASET

```
plot_history(classifier_history, 'Accuracy', 'accuracy', 'Epochs', 'Model Accuracy')
plot_history(classifier_history, 'Loss', 'loss', 'Epochs', 'Loss')
acc=efficientnet_b4_model.evaluate( test_generator, batch_size=test_batch_size, verbose=1, steps=test_steps, return_dict=False)[1]*100
print(f'accuracy on the test set is {acc:5.2f} %')
```

MEMBUAT MODEL EfficientNet-B5

```
def efficientnet_b5():
    input=Input(shape=img_shape)
    base=tf.keras.applications.EfficientNetB5(weights='imagenet',include_top=False,input_shape=img_shape,input_tensor=input,classes=7)
#     base.trainable=False
#     x=Flatten()(base.output)
    x = GlobalAveragePooling2D()(base.output)
    x = BatchNormalization()(x)
#     x = Dropout(0.2)(x)
```

```
    x = Dropout(0.5)(x)
#     x = Dense(512, activation='relu')(x)
#     x = Dropout(0.5)(x)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(7, activation='softmax',kernel_regularizer=regularizers.L1L2(l1=0.01, l2=0.01))(x)
    model = Model(input, output)
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss="categorical_crossentropy",  metrics=['accuracy'])
    return model
efficientnet_b5_model=efficientnet_b5()
```

TRAINING MODEL EfficientNet-B5

```
reduce_lr  =tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=factor,patience=5, min_lr=learning_rate,verbose=1)
checkpoint=tf.keras.callbacks.ModelCheckpoint("efficientnet_b5_model.h5",monitor="val_loss",mode="min",save_best_only = True,verbose=1)
callback_list = [ reduce_lr, checkpoint]
classifier_history = efficientnet_b5_model.fit(train_generator, batch_size=batch_size,
                  validation_data=validation_generator,
                  steps_per_epoch=train_steps,
                  validation_steps=None,
                  epochs=epoch,
                  callbacks=callback_list
)
efficientnet_b5_model.save("efficientnet_b5_model_coba.h5")
```

MEMBUAT GRAFIK EfficientNet-B5, MENGETAHUI AKURASI UNTUK TEST DATASET

```
plot_history(classifier_history, 'Accuracy', 'accuracy', 'Epochs', 'Model Accuracy')
plot_history(classifier_history, 'Loss', 'loss', 'Epochs', 'Loss')
acc=efficientnet_b5_model.evaluate( test_generator, batch_size=test_batch_size, verbose=1, steps=test_steps, return_dict=False)[1]*100
print(f'accuracy on the test set is {acc:5.2f} %')
```

MEMBUAT MODEL EfficientNet-B6

```
def efficientnet_b6():
    input=Input(shape=img_shape)
    base=tf.keras.applications.EfficientNetB6(weights='imagenet',include_top=False,input_shape=img_shape,input_tensor=input,classes=7)
#      base.trainable=False
#      x=Flatten()(base.output)
    x = GlobalAveragePooling2D()(base.output)
    x = BatchNormalization()(x)
#      x = Dropout(0.2)(x)
```

```
    x = Dropout(0.5)(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
#     x = Dense(512, activation='relu')(x)
#     x = Dropout(0.5)(x)
    output = Dense(7, activation='softmax',kernel_regularizer=regularizers.L1L2(l1=0.01, l2=0.01))(x)
    model = Model(input, output)
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss="categorical_crossentropy",  metrics=['accuracy'])
    return model
efficientnet_b6_model=efficientnet_b6()
```

TRAINING MODEL EfficientNet-B6

```
reduce_lr   =tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=factor,patience=5, min_lr=learning_rate,verbose=1)
checkpoint=tf.keras.callbacks.ModelCheckpoint("efficientnet_b6_model.h5",monitor="val_loss",mode="min",save_best_only = True,verbose=1)
callback_list = [ reduce_lr, checkpoint]
classifier_history = efficientnet_b6_model.fit(train_generator, batch_size=batch_size,
                    validation_data=validation_generator,
                    steps_per_epoch=train_steps,
                    validation_steps=None,
                    epochs=epoch,
                    callbacks=callback_list
)
efficientnet_b6_model.save("efficientnet_b6_model_coba.h5")
```

MEMBUAT GRAFIK EfficientNet-B6, MENGETAHUI AKURASI UNTUK TEST DATASET

```
plot_history(classifier_history, 'Accuracy', 'accuracy', 'Epochs', 'Model Accuracy')
plot_history(classifier_history, 'Loss', 'loss', 'Epochs', 'Loss')
acc=efficientnet_b6_model.evaluate( test_generator, batch_size=test_batch_size, verbose=1, steps=test_steps, return_dict=False)[1]*100
print(f'accuracy on the test set is {acc:5.2f} %')
```

MEMBUAT MODEL EfficientNet-B7

```
def efficientnet_b7():
    input=Input(shape=img_shape)
    base=tf.keras.applications.EfficientNetB7(weights='imagenet',include_top=False,input_shape=img_shape,input_tensor=input,classes=7)
#      base.trainable=False
#      x=Flatten()(base.output)
    x = GlobalAveragePooling2D()(base.output)
    x = BatchNormalization()(x)
#      x = Dropout(0.2)(x)
```

```
        x = Dropout(0.5)(x)
        x = Dense(512, activation='relu')(x)
        x = Dropout(0.5)(x)
        output = Dense(7, activation='softmax',kernel_regularizer=regularizers.L1L2(l1=0.01, l2=0.01))(x)
#       output = Dense(7, activation='softmax')(x)
        model = Model(input, output)
        optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
        model.compile(optimizer=optimizer, loss="categorical_crossentropy",  metrics=['accuracy'])
        return model
efficientnet_b7_model=efficientnet_b7()
```

TRAINING MODEL EfficientNet-B7

```
reduce_lr   =tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=factor,patience=5, min_lr=learning_rate,verbose=1)
checkpoint=tf.keras.callbacks.ModelCheckpoint("efficientnet_b7_model.h5",monitor="val_loss",mode="min",save_best_only = True,verbose=1)
callback_list = [ reduce_lr, checkpoint]
classifier_history = efficientnet_b7_model.fit(train_generator, batch_size=batch_size,
                    validation_data=validation_generator,
                    steps_per_epoch=train_steps,
                    validation_steps=None,
                    epochs=epoch,
                    callbacks=callback_list
)
efficientnet_b7_model.save("efficientnet_b7_model_coba.h5")
```

MEMBUAT GRAFIK EfficientNet-B7, MENGETAHUI AKURASI UNTUK TEST DATASET

```
plot_history(classifier_history, 'Accuracy', 'accuracy', 'Epochs', 'Model Accuracy')
plot_history(classifier_history, 'Loss', 'loss', 'Epochs', 'Loss')
acc=efficientnet_b7_model.evaluate( test_generator, batch_size=test_batch_size, verbose=1, steps=test_steps, return_dict=False)[1]*100
print(f'accuracy on the test set is {acc:5.2f} %')
```

LOAD MODEL DAN MENCOBA MEMPREDIKSI DENGAN DATA BERBEDA

```
from keras.models import load_model
df = pd.read_csv('/kaggle/input/ham1000-segmentation-and-classification/GroundTruth.csv')
test=df.sample(10)
classes={'AKIEC': 0, 'BCC': 1, 'BKL': 2, 'DF': 3, 'MEL': 4, 'NV': 5, 'VASC': 6}
model = load_model('/kaggle/input/efficientnet-b0-h5/efficientnet_b0_model.h5')
optimizer = tf.keras.optimizers.Adam()
# model.compile(optimizer=optimizer,loss="categorical_crossentropy", metrics=['accuracy'])
```

```
for i in test.image:
    img=preprocessing(i)
    img = np.expand_dims(img, axis=0)
    pred=model.predict(img,verbose=1)
    pred_class = np.argmax(pred)
    for clas,id in classes.items():
        if id ==pred_class:
            print(clas)
```