

SKIN BURN IMAGE DETECTION AND CLASSIFICATION

END TERM VII SEMESTER SYNOPSIS REPORT

Submitted in partial fulfilment of the requirement of the degree of

BACHELORS OF TECHNOLOGY

to

The NorthCap University

by

Lakshita 19csu166, Madhur 19csu169,

Mitali 19csu177, Nancy 19csu403

Under the supervision of

Dr. Joyanta (Industry Mentor) and Dr. Mrinal Pandey (Faculty Mentor)



Department of Computer Science and Engineering

School of Engineering and Technology

The NorthCap University, Gurugram- 122001, India

Session 2021-22

CERTIFICATE

This is to certify that the Project Synopsis entitled, “SKIN BURN IMAGE DETECTION AND CLASSIFICATION” submitted by “Lakshita Gaur 19csu166, Madhur Nagrath 19csu169, Mitali Sehgal 19csu177, Nancy Jangid 19csu403” to **The NorthCap University, Gurugram, India**, is a record of bona fide synopsis work carried out by them under my supervision and guidance and is worthy of consideration for the partial fulfilment of the degree of **Bachelor of Technology** in **Computer Science and Engineering** of the University.

<Signature of supervisor>

<Name and designation of supervisor>

Date:06/06/2022.....

INDEX

1	ABSTRACT	4
2	INTRODUCTION	4
3	BACKGROUND AND RELATED WORKS	5
4	STUDY AND COMPARISON WITH EXISTING SOLUTION.....	6
5	PROBLEM STATEMENT	9
6	OBJECTIVE	9
7	TOOLS AND PLATFORM USED	9
7.1	PROGRAMMING LANGUAGE	9
8	DESIGN METHODOLOGY	9
9	METHODOLOGY AND IMPLEMENTATION	11
9.1	Data Collection	11
9.2	MODELS IMPLEMENTED.....	12
9.2.1	Basic CNN model:	12
9.3	Resnet50:.....	17
9.3.1	Comparison	18
9.3.2	ResNet50 Architecture.....	19
9.3.3	Result	20
9.4	VGG 19:.....	21
9.4.1	Architecture.....	21
9.4.2	Results.....	23
9.4.3	Uses of the VGG Neural Network	23
9.5	INCEPTION:	23
9.5.1	Inception v3 Architecture.....	23
9.5.2	Inception v3 Training and Results	26
10	CONCLUSION	26
11	FUTURE WORK.....	27
12	REFERENCE.....	27
13	Annexure I: Responsibility Chart	28
14	Annexure II: Screenshots of all the MS-Team meetings (online)/ handwritten comments(offline) from guide.....	29
15	Annexure III: Sample of dataset	32

1 ABSTRACT

In recent years, Convolutional Neural Network (CNN) model is the state of art model successful for image analysis. In this research, we suggest integrating CNN model for burn images recognition, one kind of medical images. The aim of this research is to build an automated computer aided for identifying the degrees of burn images. The burn images dataset has been collected from Burn faculty of Cho Ray hospital, Vietnam and published by the collaboration research project of Cho Ray doctors, and the Information Technology lab of Ho Chi Minh University of Pedagogy. The CNN model was adapted to be an automated method of burn images classification into 3 degrees following the classification of burning patients in Cho Ray hospital. Skin burns in colour images must be accurately detected and classified according to burn degree in order to assist clinicians during diagnosis and early treatment. Especially in emergency cases in which clinical experience might not be available to conduct a thorough examination with high accuracy, an automated assessment may benefit patient outcomes. This burns analysis will be helpful for remote hospital in Vietnam where the hospital service must be improved. The remote doctors could use this computer aided module to classify the degrees of burns, and give the suitable medical decision.

2 INTRODUCTION

Burn injuries are the fourth leading cause of death among accidental deaths, which are themselves the third leading cause of death globally. Early and proper treatment is required to stop the degeneration of the organs and allow a full recovery. However, professionals often misguide these early treatments due to the stress and exigency of emergency situations. This phenomenon has prompted the development of artificial medical assistants, which are used to track the physicians' procedures.

Successful and proper treatment is based on an accurate analysis of injury intensity and the cause of the burn. Burn area, depth, and location are the determining factors for the injury intensity. There are three different degrees of burns, determined by the number of skin layers damaged: superficial (first degree burn), deep partial thickness (second degree burn) and full thickness (third degree burn).

In addition to massive local tissue damage, burn injuries can also provoke an inflammatory response over the whole body. Thus, patients experiencing severe burns should receive a systematic treatment applied to the patient's respiration and circulation.

The aim of skin burn images recognition is how to automatically identify the degree of burn from skin burn images of patients.

In Vietnam, applying machine learning for Image Recognition got some success such as some face detection/ recognition applications, many fingerprint recognition system. Medical Burn Images recognition is still in the first researching stage.

There are a few of research about skin image processing such as skin burn classification methods using Template Matching (TM), K nearest neighbor classifier (KNN) and Support Vector Machine (SVM), segmentation and classification of burn images using color and texture feature. Recently, CNN is the state of art for image feature extraction, image classification, image retrieval, and image recognition. This technology got success in the general images, and on some specific images.

Thus this research will customize CNN techniques like ResNet50, VGG16, VGG19 and InceptionNet to solve skin burn images recognition based on the suggestion from burning doctor in Cho Ray hospital, Vietnam. Let it called Burn Convolutional Neural Network (B-CNN).

3 BACKGROUND AND RELATED WORKS

In diagnosing clinical burning, patient need to be classified into 3 degrees of burn based on the skin's thickness, the depth of burns and scalds, and some diagnosis related groups. The degree of kin burn images recognition is a category of image recognition. Therefore, we can apply computer vision technique integrated machine learning for burn image recognition. The computer vision technique can be used in the image acquisition, and pre-processing, while the machine learning can be used in the training/recognition phase.

Serrano, Carmen, and Laura M. Roa proposed a burn images pre-processing, and segmentation using color and texture features with Euclidean distances on color space (L u v). These features are suitable input for Fuzzy Neural Network. It classifies burn into three types: superficial dermal, deep dermal and full thickness. They are different with 4 types of burn in Vietnam hospital classification system. Besides, Euclidean distances are not reflecting in the color analysis of doctor.

Suvarna, Malini, and U. C. Niranjan tried to use some classifier such as TM, k-NN, SVM work as an automatic skin burn wound analyzer. After that a suitable burn image classification using OneClass SVM has been proposed to degree of burn image recognition because of unbalance pre-labeled burn images data in Vietnam.

Recently, CNN is the current trend of image recognition in the world. CNN is not only success on some general types of images such as natural scenes, human, visual, but also able to use in medical

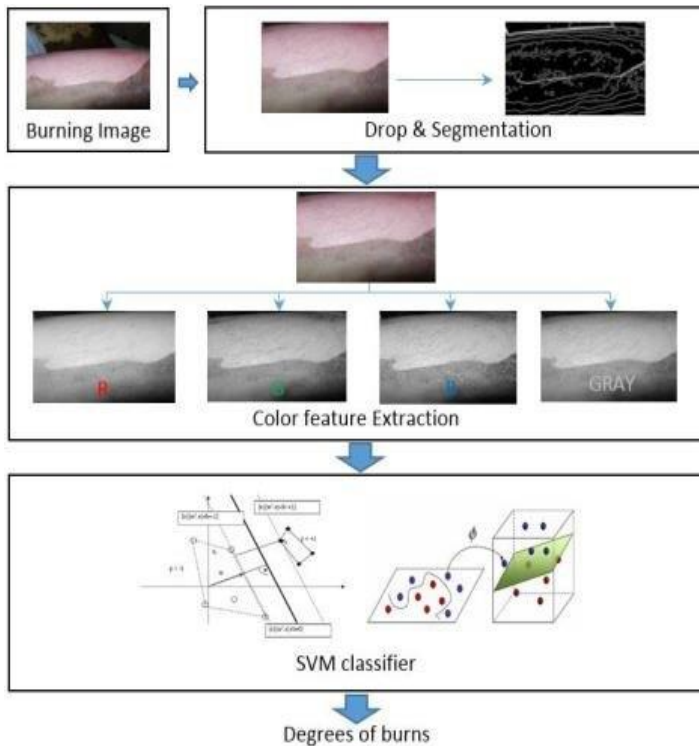
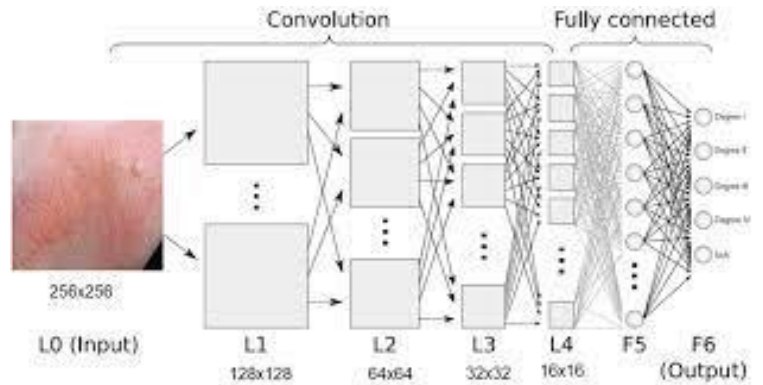


image. That is the reason why we investigate to apply CNN for Vietnam skin burn image recognition problem.



4 STUDY AND COMPARISON WITH EXISTING SOLUTION

Use of machine learning algorithms in solving real world problems is widely applied in different domains. Specifically, deep learning technique has recently achieved remarkable success in different areas such as security, traffic forecast, agriculture as well as age, gender and face recognition. Moreover, in the health sector, deep learning has been applied for image classification and has performed extremely well for detection of diseases.

A Burns is one of those traumatic injuries subjecting thousands to physical deformities, and in extreme cases, loss of lives affect different body parts such as the face, lower and upper limbs, and neck. The devastating effect is felt severely and causes discomfort to both victims, their families and to the nation as a whole. Recently, a number of researchers have attempted to address burn assessment challenges using machine learning algorithms.

A study by proposed an automation process for the identification and classification of scald burns into different categories (depth) using colour and texture features from LAB images. The experiment was conducted on 50 images in each burn depth category using K-Nearest Neighbour (KNN) and support vector machines (SVM) for the classification. The study shows SVM achieved the highest classification accuracy with 85% in first degree burns as compared to 70% by KNN,

87.5% for SVM in second degree burns as compared to 82% by KNN and 92.5% in third degree burns as compared to 75% by KNN.

Study by Suvarna, M., Toney, G., & Swastik, G. (2017) (Classification of scalding burn using image processing methods. In *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, (pp. 1312–1315)) used off-the-shelf features extracted by a pre-trained Convolutional Neural Network model and SVM as the classification algorithm for the identification of whether an image contains burns or is healthy. 1360 RGB Caucasian images (equally distributed into burnt and healthy skin) in the proposed study comprising of burn injuries from different body locations or parts. The study achieved a classification accuracy of 99.5% on Caucasian datasets

Another study by the authors in [Yadav, D., Sharma, A., Singh, M., & Goyal, A. (2019). Feature Extraction Based Machine Learning for Human Burn Diagnosis From Burn Images. *IEEE Journal of Translational Engineering in Health and Medicine*, 7, 1–7.] used 74 burn images in LAB colour space from Caucasian patients to discriminate burns using machine learning. The approach used handcrafted features to train a SVM which achieved a classification accuracy of 82.43%.

Additionally, the study reported in [Abubakar, A., Ugail, H., & Bukar, A. M. (2019) Can machine learning be used to discriminate between burns and pressure ulcer? In *Proceedings of SAI Intelligent Systems Conference*, (pp. 870–880)] proposed a work to classify burns and pressure ulcer wounds in Caucasians. Three different ImageNet pre-trained convolutional neural network models were used as feature extractors while in all the three cases SVM was used as a classifier. The evaluation was carried out via the use of tenfold cross-validation in order to avoid biases which might arise during data splitting process. Interestingly, up to 99% accuracy was recorded via the use of all the features from the ImageNet models.

The study reported was similarly based on the use of Convolutional Neural Network to predict burn depth in pediatric patients. The study was conducted on 23 burn images based on ground truth defined by experienced clinicians. Original images were then augmented via extraction of a region of interest and resulted to 676 samples out of which 119 are superficial burns, 120 are superficial partial thickness, 108 are intermediate partial thickness, 111 are deep partial thickness, 111 are normal skin images and 107 background images. The authors fine-tuned four different Convolutional Neural Network models were trained via fine-tuning; ResNet101 yields the maximum performance accuracy of 81.66%, ResNet50 yields 77.79%, GoogleNet yields 73.89%, and VGG-16 achieved 77.53%.

Towards the end, our approach in this study is using state-of-the-art deep learning model (specifically, the pre-trained ImageNet model) to discriminate burns using embedded features representations from diverse ethnicities. To the best of our knowledge, this is the first study that provides extensive experiments and analysis of classifying burns in different ethnic or racial groups.

The diagram illustrates the VGG-16 architecture, which is a deep convolutional neural network. It starts with an input image, which is processed by a 7x7 convolution with 64 filters, followed by a pooling operation. The network then consists of several stages of 3x3 convolutions with varying filter counts (64, 128, 256, 512) and pooling operations. The final output is a fully connected layer with 1000 units.

```

graph TD
    image --> conv1[7x7 conv, 64, /2]
    conv1 --> pool1[pool, /2]
    pool1 --> conv2_1[3x3 conv, 64]
    conv2_1 --> conv2_2[3x3 conv, 64]
    conv2_2 --> conv2_3[3x3 conv, 64]
    conv2_3 --> conv2_4[3x3 conv, 64]
    conv2_4 --> conv2_5[3x3 conv, 64]
    conv2_5 --> conv2_6[3x3 conv, 64]
    conv2_6 --> pool2[3x3 conv, 128, /2]
    pool2 --> conv3_1[3x3 conv, 128]
    conv3_1 --> conv3_2[3x3 conv, 128]
    conv3_2 --> conv3_3[3x3 conv, 128]
    conv3_3 --> conv3_4[3x3 conv, 128]
    conv3_4 --> conv3_5[3x3 conv, 128]
    conv3_5 --> conv3_6[3x3 conv, 128]
    conv3_6 --> conv3_7[3x3 conv, 128]
    conv3_7 --> conv3_8[3x3 conv, 128]
    conv3_8 --> conv3_9[3x3 conv, 128]
    conv3_9 --> conv3_10[3x3 conv, 128]
    conv3_10 --> conv3_11[3x3 conv, 128]
    conv3_11 --> conv3_12[3x3 conv, 128]
    conv3_12 --> conv3_13[3x3 conv, 128]
    conv3_13 --> conv3_14[3x3 conv, 128]
    conv3_14 --> conv3_15[3x3 conv, 128]
    conv3_15 --> conv3_16[3x3 conv, 128]
    conv3_16 --> conv3_17[3x3 conv, 128]
    conv3_17 --> conv3_18[3x3 conv, 128]
    conv3_18 --> conv3_19[3x3 conv, 128]
    conv3_19 --> conv3_20[3x3 conv, 128]
    conv3_20 --> conv3_21[3x3 conv, 128]
    conv3_21 --> conv3_22[3x3 conv, 128]
    conv3_22 --> conv3_23[3x3 conv, 128]
    conv3_23 --> conv3_24[3x3 conv, 128]
    conv3_24 --> conv3_25[3x3 conv, 128]
    conv3_25 --> conv3_26[3x3 conv, 128]
    conv3_26 --> conv3_27[3x3 conv, 128]
    conv3_27 --> conv3_28[3x3 conv, 128]
    conv3_28 --> conv3_29[3x3 conv, 128]
    conv3_29 --> conv3_30[3x3 conv, 128]
    conv3_30 --> conv3_31[3x3 conv, 128]
    conv3_31 --> conv3_32[3x3 conv, 128]
    conv3_32 --> conv3_33[3x3 conv, 128]
    conv3_33 --> conv3_34[3x3 conv, 128]
    conv3_34 --> conv3_35[3x3 conv, 128]
    conv3_35 --> conv3_36[3x3 conv, 128]
    conv3_36 --> conv3_37[3x3 conv, 128]
    conv3_37 --> conv3_38[3x3 conv, 128]
    conv3_38 --> conv3_39[3x3 conv, 128]
    conv3_39 --> conv3_40[3x3 conv, 128]
    conv3_40 --> conv3_41[3x3 conv, 128]
    conv3_41 --> conv3_42[3x3 conv, 128]
    conv3_42 --> conv3_43[3x3 conv, 128]
    conv3_43 --> conv3_44[3x3 conv, 128]
    conv3_44 --> conv3_45[3x3 conv, 128]
    conv3_45 --> conv3_46[3x3 conv, 128]
    conv3_46 --> conv3_47[3x3 conv, 128]
    conv3_47 --> conv3_48[3x3 conv, 128]
    conv3_48 --> conv3_49[3x3 conv, 128]
    conv3_49 --> conv3_50[3x3 conv, 128]
    conv3_50 --> conv3_51[3x3 conv, 128]
    conv3_51 --> conv3_52[3x3 conv, 128]
    conv3_52 --> conv3_53[3x3 conv, 128]
    conv3_53 --> conv3_54[3x3 conv, 128]
    conv3_54 --> conv3_55[3x3 conv, 128]
    conv3_55 --> conv3_56[3x3 conv, 128]
    conv3_56 --> conv3_57[3x3 conv, 128]
    conv3_57 --> conv3_58[3x3 conv, 128]
    conv3_58 --> conv3_59[3x3 conv, 128]
    conv3_59 --> conv3_60[3x3 conv, 128]
    conv3_60 --> conv3_61[3x3 conv, 128]
    conv3_61 --> conv3_62[3x3 conv, 128]
    conv3_62 --> conv3_63[3x3 conv, 128]
    conv3_63 --> conv3_64[3x3 conv, 128]
    conv3_64 --> conv3_65[3x3 conv, 128]
    conv3_65 --> conv3_66[3x3 conv, 128]
    conv3_66 --> conv3_67[3x3 conv, 128]
    conv3_67 --> conv3_68[3x3 conv, 128]
    conv3_68 --> conv3_69[3x3 conv, 128]
    conv3_69 --> conv3_70[3x3 conv, 128]
    conv3_70 --> conv3_71[3x3 conv, 128]
    conv3_71 --> conv3_72[3x3 conv, 128]
    conv3_72 --> conv3_73[3x3 conv, 128]
    conv3_73 --> conv3_74[3x3 conv, 128]
    conv3_74 --> conv3_75[3x3 conv, 128]
    conv3_75 --> conv3_76[3x3 conv, 128]
    conv3_76 --> conv3_77[3x3 conv, 128]
    conv3_77 --> conv3_78[3x3 conv, 128]
    conv3_78 --> conv3_79[3x3 conv, 128]
    conv3_79 --> conv3_80[3x3 conv, 128]
    conv3_80 --> conv3_81[3x3 conv, 128]
    conv3_81 --> conv3_82[3x3 conv, 128]
    conv3_82 --> conv3_83[3x3 conv, 128]
    conv3_83 --> conv3_84[3x3 conv, 128]
    conv3_84 --> conv3_85[3x3 conv, 128]
    conv3_85 --> conv3_86[3x3 conv, 128]
    conv3_86 --> conv3_87[3x3 conv, 128]
    conv3_87 --> conv3_88[3x3 conv, 128]
    conv3_88 --> conv3_89[3x3 conv, 128]
    conv3_89 --> conv3_90[3x3 conv, 128]
    conv3_90 --> conv3_91[3x3 conv, 128]
    conv3_91 --> conv3_92[3x3 conv, 128]
    conv3_92 --> conv3_93[3x3 conv, 128]
    conv3_93 --> conv3_94[3x3 conv, 128]
    conv3_94 --> conv3_95[3x3 conv, 128]
    conv3_95 --> conv3_96[3x3 conv, 128]
    conv3_96 --> conv3_97[3x3 conv, 128]
    conv3_97 --> conv3_98[3x3 conv, 128]
    conv3_98 --> conv3_99[3x3 conv, 128]
    conv3_99 --> conv3_100[3x3 conv, 128]
    conv3_100 --> conv3_101[3x3 conv, 128]
    conv3_101 --> conv3_102[3x3 conv, 128]
    conv3_102 --> conv3_103[3x3 conv, 128]
    conv3_103 --> conv3_104[3x3 conv, 128]
    conv3_104 --> conv3_105[3x3 conv, 128]
    conv3_105 --> conv3_106[3x3 conv, 128]
    conv3_106 --> conv3_107[3x3 conv, 128]
    conv3_107 --> conv3_108[3x3 conv, 128]
    conv3_108 --> conv3_109[3x3 conv, 128]
    conv3_109 --> conv3_110[3x3 conv, 128]
    conv3_110 --> conv3_111[3x3 conv, 128]
    conv3_111 --> conv3_112[3x3 conv, 128]
    conv3_112 --> conv3_113[3x3 conv, 128]
    conv3_113 --> conv3_114[3x3 conv, 128]
    conv3_114 --> conv3_115[3x3 conv, 128]
    conv3_115 --> conv3_116[3x3 conv, 128]
    conv3_116 --> conv3_117[3x3 conv, 128]
    conv3_117 --> conv3_118[3x3 conv, 128]
    conv3_118 --> conv3_119[3x3 conv, 128]
    conv3_119 --> conv3_120[3x3 conv, 128]
    conv3_120 --> conv3_121[3x3 conv, 128]
    conv3_121 --> conv3_122[3x3 conv, 128]
    conv3_122 --> conv3_123[3x3 conv, 128]
    conv3_123 --> conv3_124[3x3 conv, 128]
    conv3_124 --> conv3_125[3x3 conv, 128]
    conv3_125 --> conv3_126[3x3 conv, 128]
    conv3_126 --> conv3_127[3x3 conv, 128]
    conv3_127 --> conv3_128[3x3 conv, 128]
    conv3_128 --> conv3_129[3x3 conv, 128]
    conv3_129 --> conv3_130[3x3 conv, 128]
    conv3_130 --> conv3_131[3x3 conv, 128]
    conv3_131 --> conv3_132[3x3 conv, 128]
    conv3_132 --> conv3_133[3x3 conv, 128]
    conv3_133 --> conv3_134[3x3 conv, 128]
    conv3_134 --> conv3_135[3x3 conv, 128]
    conv3_135 --> conv3_136[3x3 conv, 128]
    conv3_136 --> conv3_137[3x3 conv, 1
```


5 PROBLEM STATEMENT

Accurate assessment of burns is increasingly sought due to diagnostic challenges faced with traditional visual assessment methods. While visual assessment is the most established means of evaluating burns globally, specialised dermatologists are not readily available in most locations and assessment is highly subjective. The use of other technical devices such as Laser Doppler Imaging is highly expensive while rate of occurrences is high in low- and middle-income countries. These necessitate the need for robust and cost-effective assessment techniques thereby acting as an affordable alternative to human expertise.

6 OBJECTIVE

Our objective is to present a technique to discriminate skin burns using deep transfer learning. This is due to deficient datasets to train a model from scratch, in which two dense and a classification layers were added to replace the existing top layers of pre-trained ResNet50 model.

7 TOOLS AND PLATFORM USED

Google Colabs

7.1 PROGRAMMING LANGUAGE

Python Lib: Tensorflow, Keras, Numpy, Pandas, Seaborn and Matplotlib.

8 DESIGN METHODOLOGY

Convolutional Neural Networks (ConvNet) are machine learning algorithms inspired by the human brain and are used as architecture for classification such as image recognition. ConvNet architecture generally consists of a series of different layers, where in each layer 2D array of pixels (feature maps) are produced which serves an input to the next layer. Training of ConvNet architecture was a bottleneck to researchers due to inability to access a huge amount of data and powerful computational machines until around 2010 when a large repository of images called ImageNet was made available. The ConvNet architecture is fundamentally made up of the following layers :

- **Input layer** this is where data are passed into the network. The data can be a raw image pixel or their transformations.
- **Convolutional layer** this layer contains fixed size filters arranged in series performing convolution operations and producing what is referred to as a feature map.
- **Pooling layer** this is where the dimensions of the feature map produced by convolutional layer are reduced, thereby allowing the network to focused on the most important features.

- **Rectified Linear Unit (ReLU)** this layer is responsible for removing all negative values by applying a non-linear function to the output of the previous layer and setting them to zero.
- **Fully connected Layer** this is the layer where the high-level reasoning of the patterns generated by the previous layers is done. All the activations in the previous layer have full connection to neurons in this layer. For feature extraction using pre-trained ConvNet model, features are generated here and used to train another classification algorithm
- **Loss layer** this is where the deviation between the true and the predicted labels is penalized. This is normally the last layer of the ConvNet, and various loss functions are used depending on the task. Example of loss functions includes SoftMax, Cross-Entropy and Sigmoid

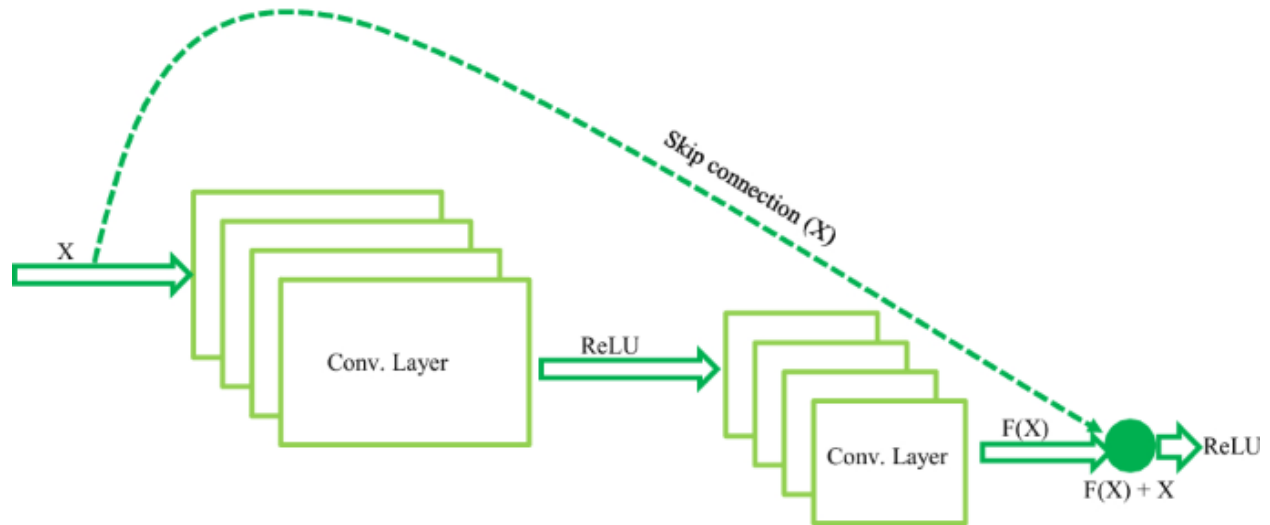
Deep ConvNet models become popular and the research domain receive more recognition due data availability and the computational resources from 2010. Among the most common ConvNet models from 2010 to date includes AlexNet, VGG-16 and ResNet-50. AlexNet model is composed of 5 convolutional layers with an interweaved max-pooling layers and 3 fully-connected layers proposed by the authors of from the University of Toronto in Canada. The first convolutional layer was configured to take an input size 224×224 and equipped with 96 filters of size 11×11 with a stride of 4. The output of the first layers goes into the second layer as input (after the pooling operation) equipped with 256 filters of size 5×5 . The third, fourth and fifth layers contain 384, 384 and 256 filters respectively with the same filter size of 3×3 while the fully connected layers have 4096 neurons each and the soft-max layer as the final layer for classification.

In 2014, Visual Geometry Group (VGG) from Oxford University made another breakthrough in image classification task using a sixteen layered ConvNet model; thirteen convolutional and three fully connected layers. Apart from being deeper than AlexNet, the size of receptive fields was significantly reduced in both convolutional and pooling layers to 3×3 and 2×2 respectively and stride of 2 was maintained throughout. Similar to AlexNet model, the soft-max layer was used as the final classification layer.

GoogleNet achieved a remarkable performance in 2014. It has a total of 22 layers or 27 layers (pooling layers inclusive). The architecture of the model comprises of parallel layers of convolution with different filters, the output of such parallel convolutional layers is then concatenated as input to subsequent layer. Unlike previously proposed models, GoogleNet is also equipped with 1×1 convolution for dimensionality reduction. GoogleNet outperformed all other proposed models in the ILSVRC competition in 2014.

However, deeper ConvNet encountered several difficulties during training which includes vanishing gradients and accuracy degradation, which was one of the issues faced by VGG. When the network is deep, the gradient shrinks to zero from where the loss function was computed, thereby resulting in a network not learning anything. As such, among the researchers who proposed a pipeline to address the challenges by allowing the network to go deeper with the increase in performance includes Residual Network from Microsoft, and won the ILSVRC competition in 2015. Residual Network (known as ResNet) uses skip connections to allow a copy of the gradient

to be passed to the subsequent layer without passing through other weight layers as depicted in Fig. below



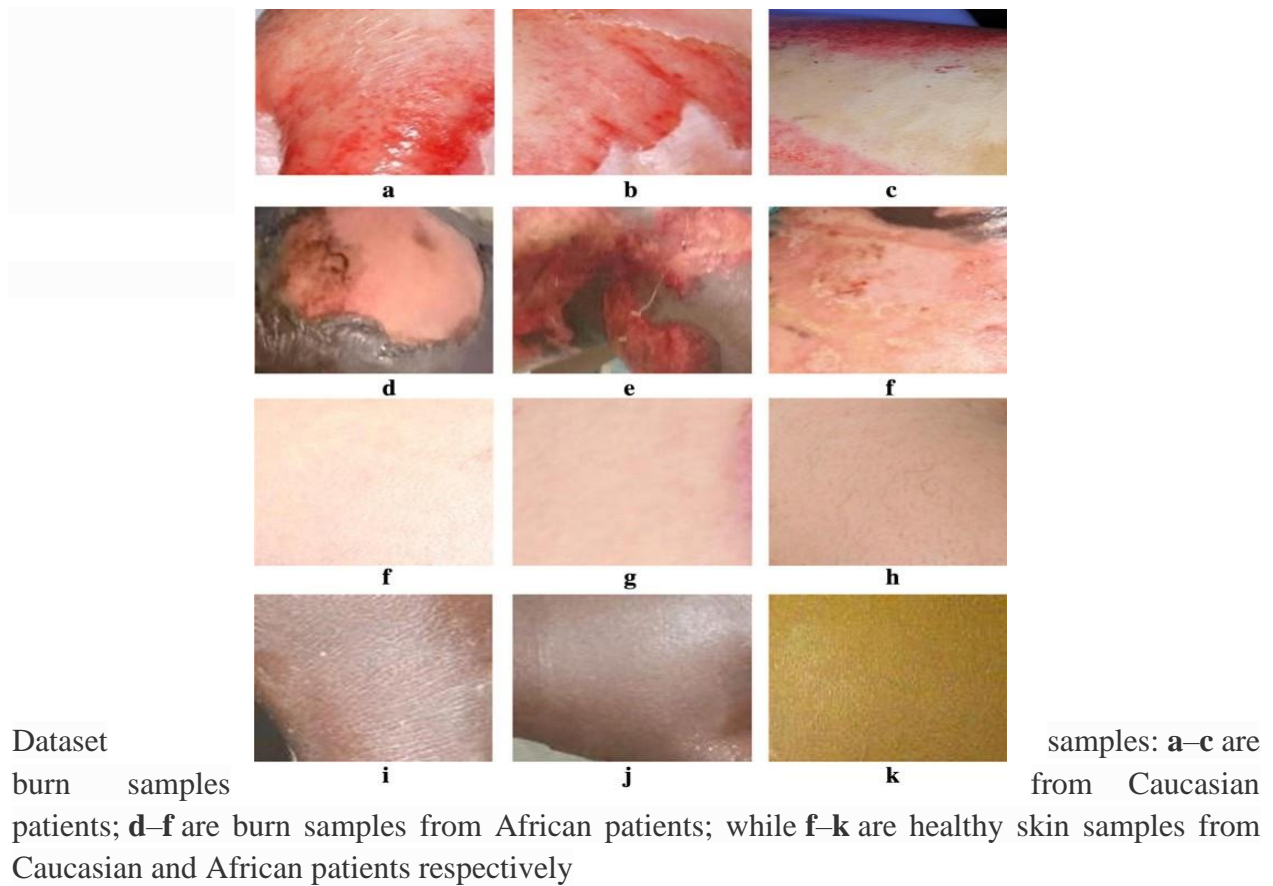
Towards the end, our proposal in our project is straight-forward, which involves automatic recognition of burns using deep learning.

9 METHODOLOGY AND IMPLEMENTATION

Generally, there are several existing pre-trained models that are used for transfer learning. For the problem domain, we propose in this study, and we utilised VGG16, VGG19, InceptionV3, ResNet50 pre-trained model.

9.1 Data Collection

Datasets were collected from patients of different ethnicities involving Caucasians, Vietnam and Africans. The Caucasian datasets were collected in Bradford hospital, the United Kingdom and the African datasets were collected from Federal Teaching Hospital Gombe, Nigeria. 1360 Caucasian images were successfully collected, which contained 680 burn images and 680 healthy skin images. African dataset contains 270 burn images and 270 healthy skin images, totaling 540 images. Figure below shows samples of burns from the two ethnicities. The database images are composed of both pediatric and adult patients as well as different burn complexities.



9.2 MODELS IMPLEMENTED

9.2.1 Basic CNN model:

Three basic components to define a basic convolutional neural network.
The Convolutional Layer, The Pooling layer, The Output layer

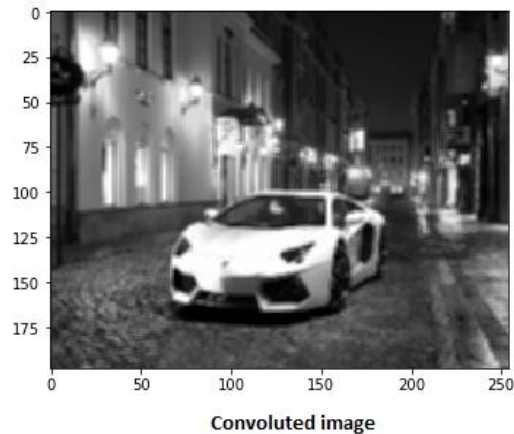
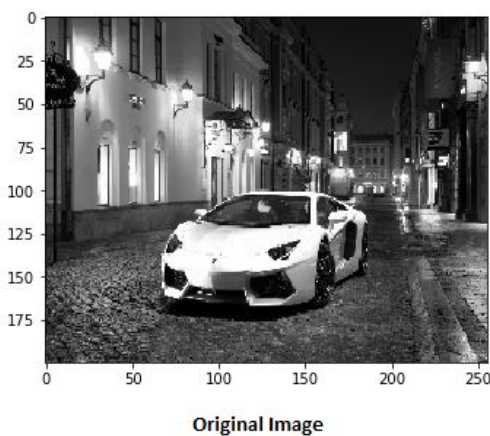
9.2.1.1 The Convolutional Layer :

INPUT IMAGE						WEIGHT			
18	54	51	239	244	188	1	0	1	429
55	121	75	78	95	88	0	1	0	
35	24	204	113	109	221	1	0	1	
3	154	104	235	25	130				
15	253	225	159	78	233				
68	85	180	214	245	0				

In this layer if we have an image of size 66. We define a weight matrix which extracts certain features from the images. We have initialized the weight as a 3x3 matrix. This weight shall now run across the image such that all the pixels are covered at least once, to give a convolved output. The value 429 above, is obtained by the adding the values obtained by element wise multiplication of the weight matrix and the highlighted 3x3 part of the input image.

INPUT IMAGE						WEIGHT			
18	54	51	239	244	188	1	0	1	429
55	121	75	78	95	88	0	1	0	
35	24	204	113	109	221	1	0	1	
3	154	104	235	25	130				
15	253	225	159	78	233				
68	85	180	214	245	0				

The 66 image is now converted into a 44 image. Think of weight matrix like a paint brush painting a wall. The brush first paints the wall horizontally and then comes down and paints the next row horizontally. Pixel values are used again when the weight matrix moves along the image. This basically enables parameter sharing in a convolutional neural network.



- The weight matrix behaves like a filter in an image, extracting particular information from the original image matrix.
- A weight combination might be extracting edges, while another one might extract a particular color, while another one might just blur the unwanted noise.
- The weights are learnt such that the loss function is minimized and extract features from the original image which help the network in correct prediction.
- When we use multiple convolutional layers, the initial layer extract more generic features, and as network gets deeper the features get complex.

9.2.1.2 What is Stride?

As shown above, the filter or the weight matrix we moved across the entire image moving one pixel at a time. If this is a hyperparameter to move weight matrix 1 pixel at a time across image it is called as stride of 1. Let us see for stride of 2 how it looks.

INPUT IMAGE					WEIGHT			
18	54	51	239	244	1	0	1	429
55	121	75	78	95	0	1	0	
35	24	204	113	109	1	0	1	
3	154	104	235	25				
15	253	225	159	78				

As you can see the size of image keeps on reducing as we increase the stride value.

Padding the input image with zeros across it solves this problem for us. We can also add more than one layer of zeros around the image in case of higher stride values.

0	0	0	0	0	0	0	0	0
0	18	54	51	239	244	188	0	0
0	55	121	75	78	95	88	0	0
0	35	24	204	113	109	221	0	0
0	3	154	104	235	25	130	0	0
0	15	253	225	159	78	233	0	0
0	68	85	180	214	245	0	0	0
0	0	0	0	0	0	0	0	0

We can see how the initial shape of the image is retained after we padded the image with a zero. This is known as same padding since the output image has the same size as the input.

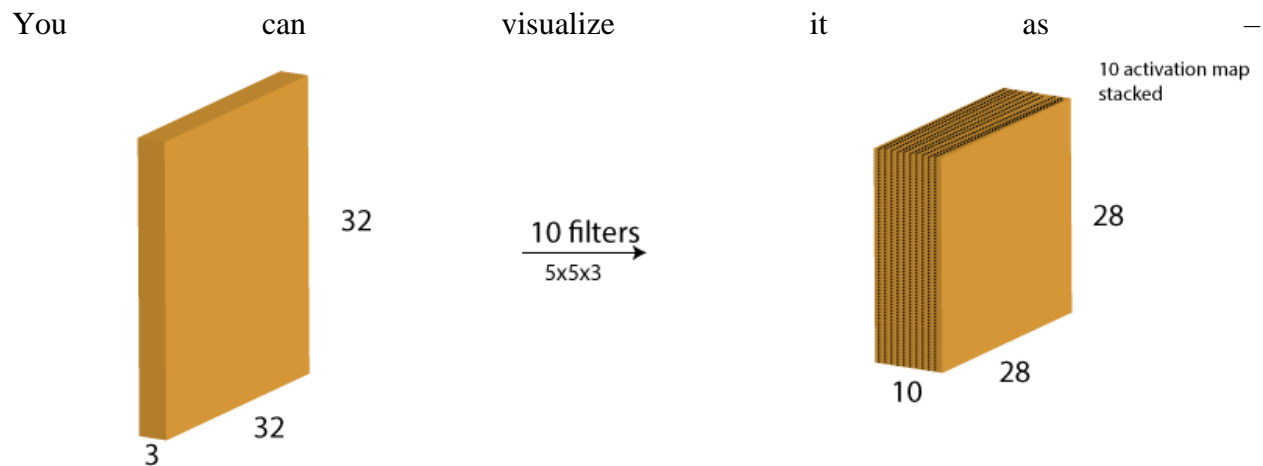
									WEIGHT			
0	0	0	0	0	0	0	0	0	1	0	1	139
0	18	54	51	239	244	188	0	0	0	1	0	
0	55	121	75	78	95	88	0	0	1	0	1	
0	35	24	204	113	109	221	0	0				
0	3	154	104	235	25	130	0	0				
0	15	253	225	159	78	233	0	0				
0	68	85	180	214	245	0	0	0				
0	0	0	0	0	0	0	0	0				

This is known as same padding (which means that we considered only the valid pixels of the input image). The middle 4*4 pixels would be the same. Here we have retained more information from the borders and have also preserved the size of the image.

9.2.1.3 Having Multiple filters & the Activation Map

- The depth dimension of the weight would be same as the depth dimension of the input image.
- The weight extends to the entire depth of the input image.
- Convolution with a single weight matrix would result into a convolved output with a single depth dimension. In case of multiple filters all have same dimensions applied together.
- The output from the each filter is stacked together forming the depth dimension of the convolved image.

Suppose we have an input image of size 32x32x3. And we apply 10 filters of size 5x5x3 with valid padding. The output would have the dimensions as 28x28x10.



This activation map is the output of the convolution layer.

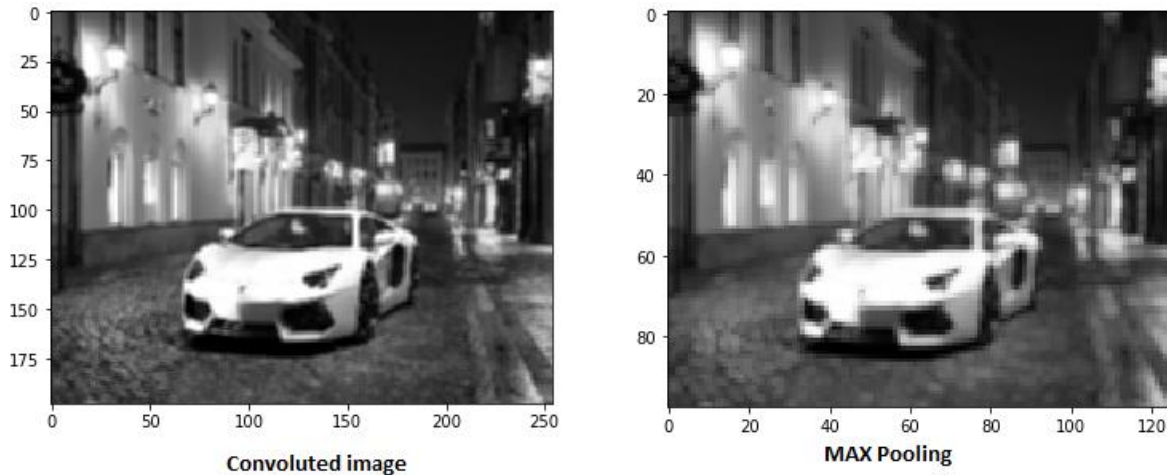
9.2.1.4 The Pooling Layer :

If images are big in size, we would need to reduce the no. of trainable parameters. For this we need to use pooling layers between convolution layers. Pooling is used for reducing the spatial size of the image and is implemented independently on each depth dimension resulting in no change in image depth. Max pooling is the most popular form of pooling layer.

429	505	686	856
261	792	412	640
633	653	851	751
608	913	713	657

792	856
913	851

Here we have taken stride as 2, while pooling size also as 2. The max operation is applied to each depth dimension of the convolved output. As you can see, the 44 convolved output has become 22 after the max pooling operation.



In the above image we have taken a convolved image and applied max pooling on it which resulted in still retaining the image information that is a car but if we closely observe the dimensions of the image is reduced to half which basically means we can reduce the parameters to a great number.

There are other forms of pooling like average pooling, L2 norm pooling.

9.2.1.5 Output dimensions

It is tricky at times to understand the input and output dimensions at the end of each convolution layer. For this we will use three hyperparameters that would control the size of output volume.

1. No of Filter: The depth of the output volume will be equal to the number of filter applied. The depth of the activation map will be equal to the number of filters.
2. Stride – When we have a stride of one we move across and down a single pixel. With higher stride values, we move large number of pixels at a time and hence produce smaller output volumes.
3. Zero padding – This helps us to preserve the size of the input image. If a single zero padding is added, a single stride filter movement would retain the size of the original image.

We can apply a simple formula to calculate the output dimensions.

The spatial size of the output image can be calculated as $([W-F+2P]/S)+1$. where, W is the input volume size, F is the size of the filter, P is the number of padding applied S is the number of strides.

Let us take an example of an input image of size 64x64x3, we apply 10 filters of size 3x3x3, with single stride and no zero padding.

Here $W=64$, $F=3$, $P=0$ and $S=1$. The output depth will be equal to the number of filters applied i.e. 10.

The size of the output volume will be $([64-3+0]/1)+1 = 62$. Therefore the output volume will be 62x62x10.

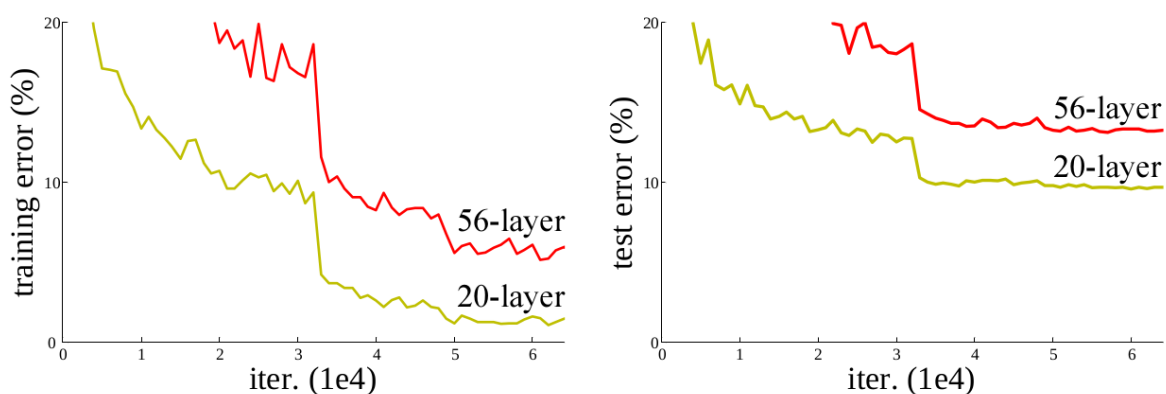
9.2.1.6 The Output layer :

- With no of layers of convolution and padding, we need the output in the form of a class.
- To generate the final output we need to apply a fully connected layer to generate an output equal to the number of classes we need.
- Convolution layers generate 3D activation maps while we just need the output as whether or not an image belongs to a particular class.
- The Output layer has a loss function like categorical cross-entropy, to compute the error in prediction. Once the forward pass is complete the backpropagation begins to update the weight and biases for error and loss reduction

9.3 Resnet50:

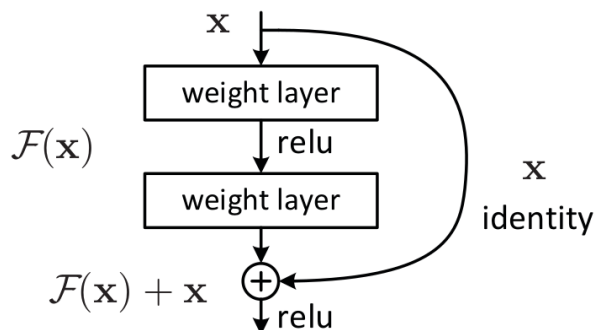
In 2012 at the LSVRC2012 classification contest AlexNet won the the first price, After that ResNet was the most interesting thing that happened to the computer vision and the deep learning world.

Because of the framework that ResNets presented it was made possible to train ultra deep neural networks that can contain hundreds or thousands of layers and still achieve great performance.



In this Figure we can see on the left and the right that the deeper model is always producing more error, where in fact it shouldn't have done that.

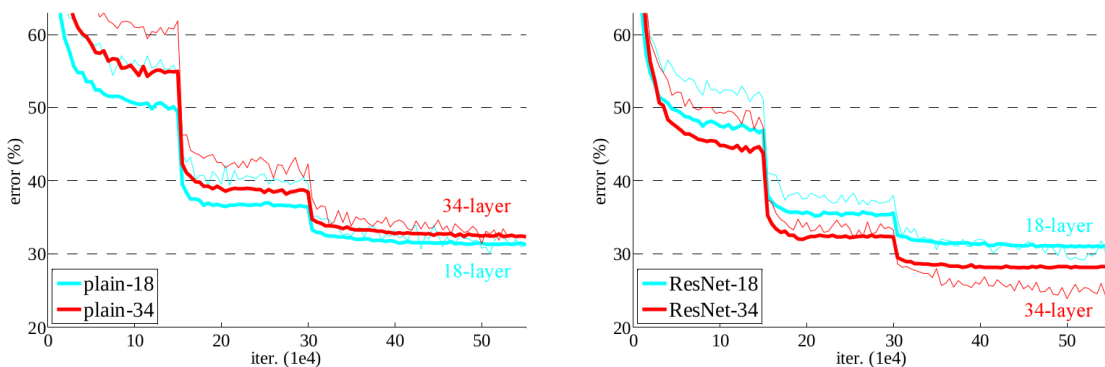
The authors addressed this problem by introducing deep residual learning framework so for this they introduce shortcut connections that simply perform identity mappings



They explicitly let the layers fit a residual mapping and denoted that as $H(x)$ and they let the non linear layers fit another mapping $F(x) := H(x) - x$ so the original mapping becomes $H(x) := F(x) + x$ as can be seen in Figure 2.

And the benefit of these shortcut identity mapping were that there was no additional parameters added to the model and also the computational time was kept in check.

9.3.1 Comparison



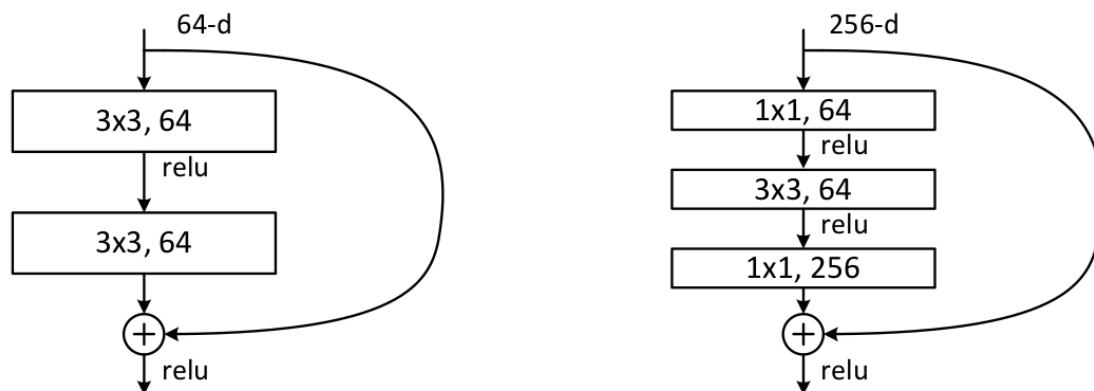
To demonstrate how much better the ResNet are they compared it with a 34 layer model and a 18 layer model both with plain and residual mappings and the results were not so astounding the 18 layer plain net outperformed the 34 layer plain net and in the case of ResNet the 34 layer ResNet outperformed the 18 layer ResNet as can be seen in figure 3.

9.3.2 ResNet50 Architecture

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Now we are going to discuss about Resnet 50 and also the architecture for the above talked 18 and 34 layer ResNet is also given residual mapping and not shown for simplicity.

There was a small change that was made for the ResNet 50 and above that before this the shortcut connections skipped two layers but now they skip three layers and also there was 1 * 1 convolution layers added that we are going to see in detail with the ResNet 50 Architecture.



So as we can see in the table 1 the resnet 50 architecture contains the following element:

- A convoution with a kernel size of 7 * 7 and 64 different kernels all with a stride of size 2 giving us **1 layer**.

- Next we see max pooling with also a stride size of 2.
- In the next convolution there is a $1 * 1,64$ kernel following this a $3 * 3,64$ kernel and at last a $1 * 1,256$ kernel, These three layers are repeated in total 3 time so giving us **9 layers** in this step.
- Next we see kernel of $1 * 1,128$ after that a kernel of $3 * 3,128$ and at last a kernel of $1 * 1,512$ this step was repeated 4 time so giving us **12 layers** in this step.
- After that there is a kernel of $1 * 1,256$ and two more kernels with $3 * 3,256$ and $1 * 1,1024$ and this is repeated 6 time giving us a total of **18 layers**.
- And then again a $1 * 1,512$ kernel with two more of $3 * 3,512$ and $1 * 1,2048$ and this was repeated 3 times giving us a total of **9 layers**.
- After that we do a average pool and end it with a fully connected layer containing 1000 nodes and at the end a softmax function so this gives us **1 layer**.

We don't actually count the activation functions and the max/ average pooling layers.

so totaling this it gives us a $1 + 9 + 12 + 18 + 9 + 1 = 50$ layers Deep Convolutional network.

9.3.3 Result

The Result were pretty good on the ImageNet validation set, The ResNet 50 model achieved a top-1 error rate of 20.47 percent and achieved a top-5 error rate of 5.25 percent, This is reported for single model that consists of 50 layers not a ensemble of it. below is the table given if you want to compare it with other ResNets or with other models.

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43 [†]
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	19.38	4.49

9.4 VGG 19:

VGG-19 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 19 layers deep and can classify images into 1000 object categories, such as a keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images.

It is a variant of the VGG model which in short consists of 19 layers (16 convolution layers, 3 Fully connected layer, 5 MaxPool layers and 1 SoftMax layer). There are other variants of VGG like VGG11, VGG16 and others. VGG19 has **19.6 billion FLOPs**.

So in simple language, VGG is a deep CNN used to classify images. The layers in the VGG19 model are as follows:

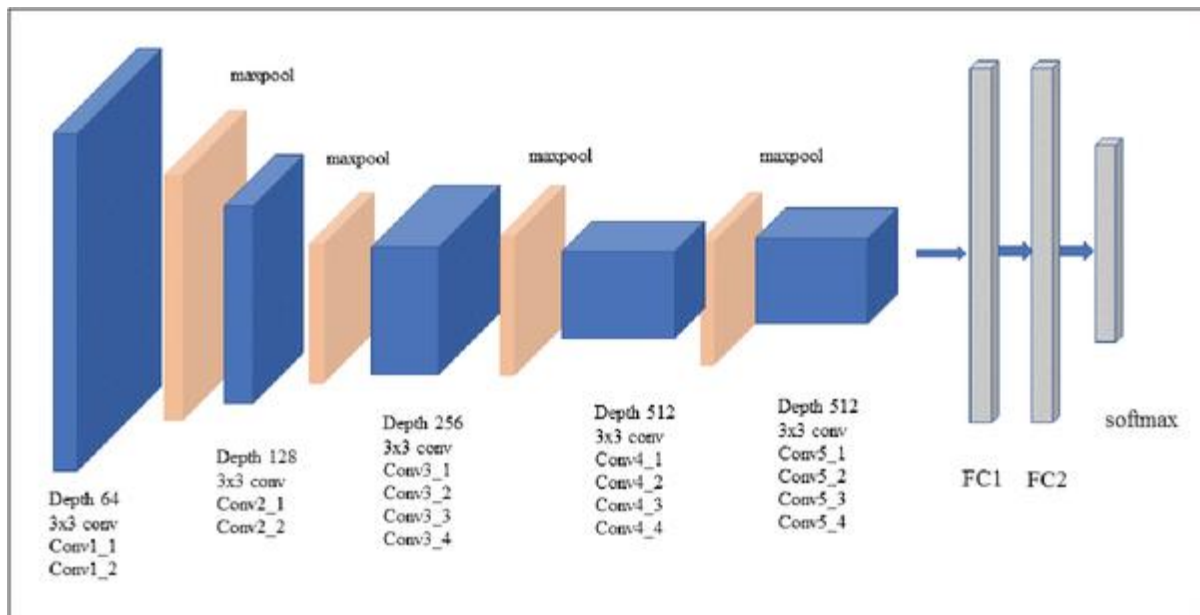


Fig. 3. VGG-19 network architecture

9.4.1 Architecture

- Fixed size of (224 * 224) RGB image was given as input to this network which means that the matrix was of shape (224,224,3).
- The only preprocessing that was done is that they subtracted the mean RGB value from each pixel, computed over the whole training set.
- Used kernels of (3 * 3) size with a stride size of 1 pixel, this enabled them to cover the whole notion of the image.
- spatial padding was used to preserve the spatial resolution of the image.
- max pooling was performed over a 2 * 2 pixel windows with stride 2.

- this was followed by Rectified linear unit(ReLU) to introduce non-linearity to make the model classify better and to improve computational time as the previous models used tanh or sigmoid functions this proved much better than those.
- implemented three fully connected layers from which the first two were of size 4096 and after that, a layer with 1000 channels for 1000-way *ILSVRC* classification and the final layer is a softmax function.

The column E in the following table is for VGG19 (other columns are for other variants of VGG models):

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

9.4.2 Results

Comparison between other state of the art models presented at *ILSVRC*.

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	23.7	6.8	6.8
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	6.7	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

9.4.3 Uses of the VGG Neural Network

The main purpose for which the VGG net was designed was to win the *ILSVRC* but it has been used in many other ways.

- Used just as a good classification architecture for many other datasets and as the authors made the models available to the public they can be used as is or with modification for other similar tasks also.
- Transfer learning: can be used for facial recognition tasks also.
- weights are easily available with other frameworks like keras so they can be tinkered with and used for as one wants.
- Content and style loss using VGG-19 network

9.5 INCEPTION:

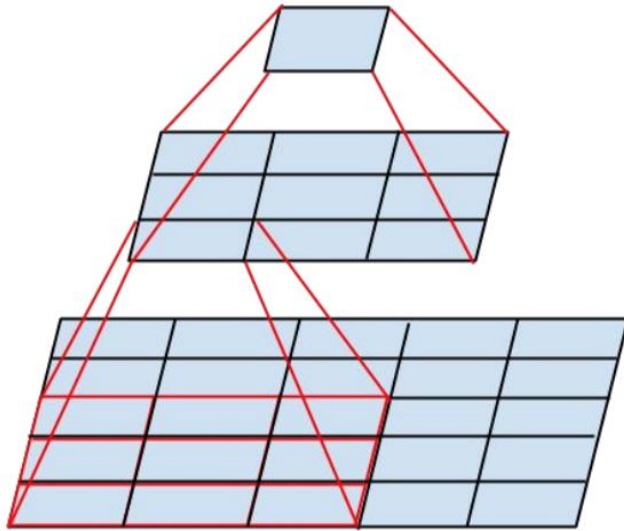
In comparison to VGGNet, Inception Networks (GoogLeNet/Inception v1) have proved to be more computationally efficient, both in terms of the number of parameters generated by the network and the economical cost incurred (memory and other resources). If any changes are to be made to an Inception Network, care needs to be taken to make sure that the computational advantages aren't lost. Thus, the adaptation of an Inception network for different use cases turns out to be a problem due to the uncertainty of the new network's efficiency. In an Inception v3 model, several techniques for optimizing the network have been put suggested to loosen the constraints for easier model adaptation. The techniques include factorized convolutions, regularization, dimension reduction, and parallelized computations.

9.5.1 Inception v3 Architecture

The architecture of an Inception v3 network is progressively built, step-by-step, as explained below:

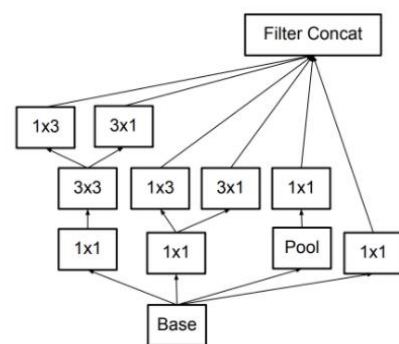
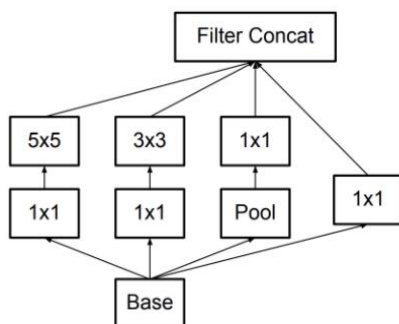
Factorized Convolutions: this helps to reduce the computational efficiency as it reduces the number of parameters involved in a network. It also keeps a check on the network efficiency.

Smaller convolutions: replacing bigger convolutions with smaller convolutions definitely leads to faster training. Say a 5×5 filter has 25 parameters; two 3×3 filters replacing a 5×5 convolution has only 18 ($3 \times 3 + 3 \times 3$) parameters instead.



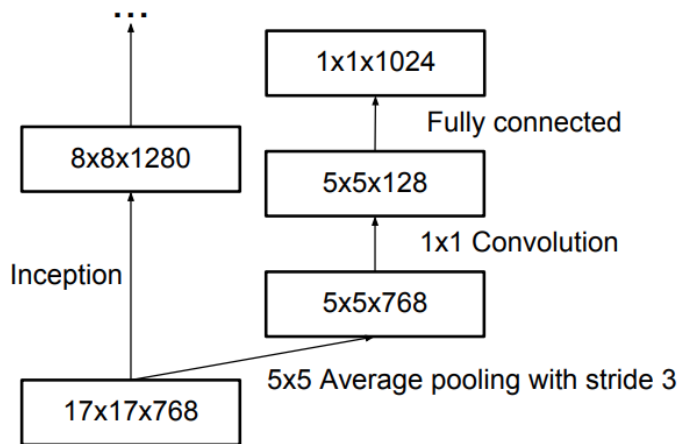
In the middle we see a 3×3 convolution, and below a fully-connected layer. Since both 3×3 convolutions can share weights among themselves, the number of computations can be reduced.

Asymmetric convolutions: A 3×3 convolution could be replaced by a 1×3 convolution followed by a 3×1 convolution. If a 3×3 convolution is replaced by a 2×2 convolution, the number of parameters would be slightly higher than the asymmetric convolution proposed.

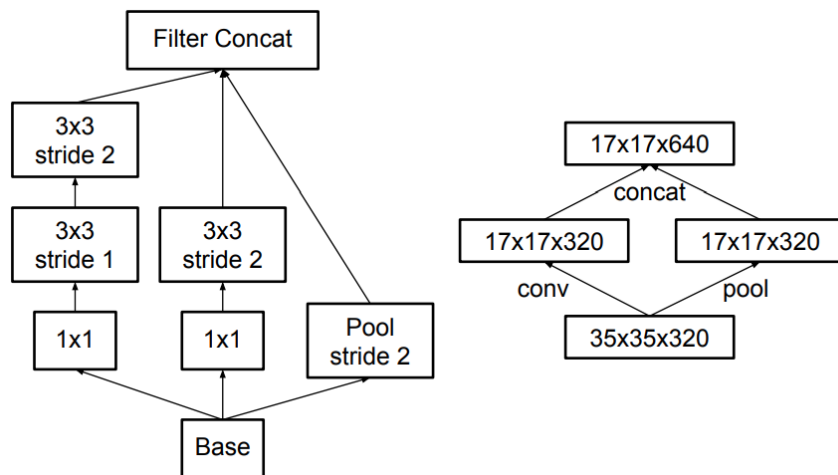


- Content and style loss using VGG-19 network
- Content and style loss using VGG-19 network

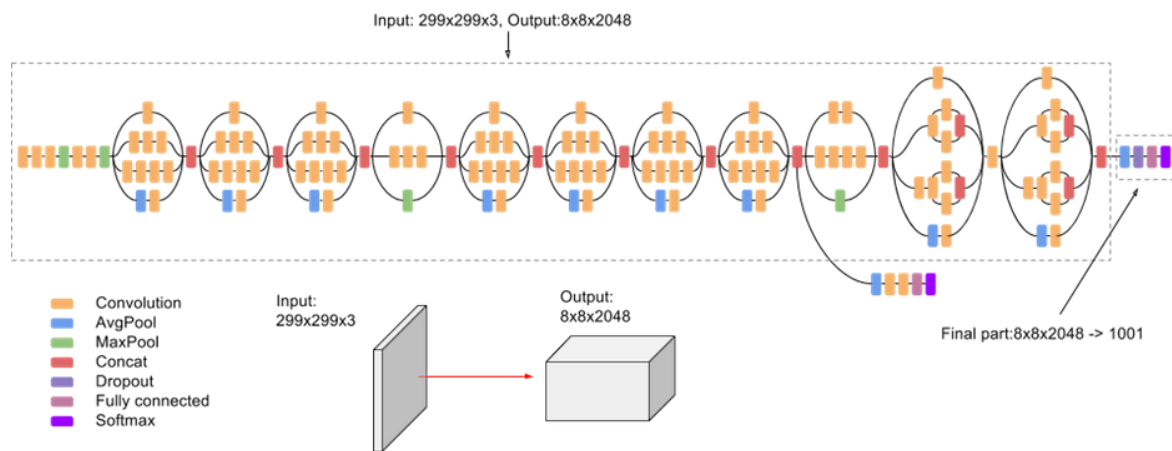
Auxiliary classifier: an auxiliary classifier is a small CNN inserted between layers during training, and the loss incurred is added to the main network loss. In GoogLeNet auxiliary classifiers were used for a deeper network, whereas in Inception v3 an auxiliary classifier acts as a regularizer.



Grid size reduction: Grid size reduction is usually done by pooling operations. However, to combat the bottlenecks of computational cost, a more efficient technique is proposed:



All the above concepts are consolidated into the final architecture.



9.5.2 Inception v3 Training and Results

Inception v3 was trained on ImageNet and compared with other contemporary models, as shown below.

Network	Top-1 Error	Top-5 Error	Cost Bn Ops
GoogLeNet [20]	29%	9.2%	1.5
BN-GoogLeNet	26.8%	-	1.5
BN-Inception [7]	25.2%	7.8	2.0
Inception-v3-basic	23.4%	-	3.8
Inception-v3-rmsprop RMSProp	23.1%	6.3	3.8
Inception-v3-smooth Label Smoothing	22.8%	6.1	3.8
Inception-v3-fact Factorized 7×7	21.6%	5.8	4.8
Inception-v3 BN-auxiliary	21.2%	5.6%	4.8

As shown in the table, when augmented with an auxiliary classifier, factorization of convolutions, RMSProp, and Label Smoothing, Inception v3 can achieve the lowest error rates compared to its contemporaries.

10 CONCLUSION

The model implementation was done through Resnet50, Basic CNN, Inception V3 and Vgg16 and 19. Images were imported after transformation and every model was run for 25 epoch values. The Vgg16 model gave the highest accuracy of 96.3%, followed by Inception Net with an accuracy of 94.8%.

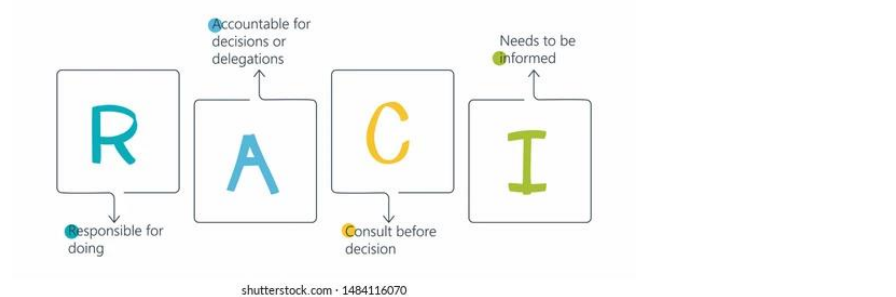
11 FUTURE WORK

- Increase accuracy of the models implemented to predict more efficient results.
- Deploy model for further usage.
- Research paper to showcase the work done.

12 REFERENCE

- [1] D.P. Yadav, A. Sharma, M. Singh, A. Goyal, Feature extraction based machine learning for human burn diagnosis from burn images, *IEEE J Transl Eng Health Med* 7 (1800507) (2019) 1–7, <https://doi.org/10.1109/JTEHM.2019.2923628>.
- [2] A.J. Singer, S.T. Boyce, Burn wound healing and tissue engineering, *J. Burn Care Res.* 38 (3) (2017) 605–613, <https://doi.org/10.1097/BCR.0000000000000538>.
- [3] K.B. Mitchell, E. Khalil, A. Brennan, H. Shao, A. Rabbitts, N.E. Leahy, R.W. Yurt, J. J. Gallagher, New management strategy for fluid resuscitation: quantifying volume in the first 48 hours after burn injury, *J. Burn Care Res.* 34 (1) (2013) 196–202, <https://doi.org/10.1097/BCR.0b013e3182700965>.
- [4] U. S ,evik, E. Karakullukçu, T. Berber, Y. Akbas , S. Türkyılmaz, Automatic classification of skin burn colour images using texture-based feature extraction, *IET Image Process.* 13 (11) (2019) 2018–2028, <https://doi.org/10.1049/iet-ipr.2018.5899>.
- [5] A.U. Rehman-Butt, W. Ahmad, R. Ashraf, M. Asif, S. Ashraf-Cheema, Computer aided diagnosis (CAD) for segmentation and classification of burnt human skin, in: *Proceedings of the 2019 International Conference on Electrical, Communication, and Computer Engineering, ICECCE*, 2019, pp. 1–5, <https://doi.org/10.1109/ICECCE47252.2019.8940758>.

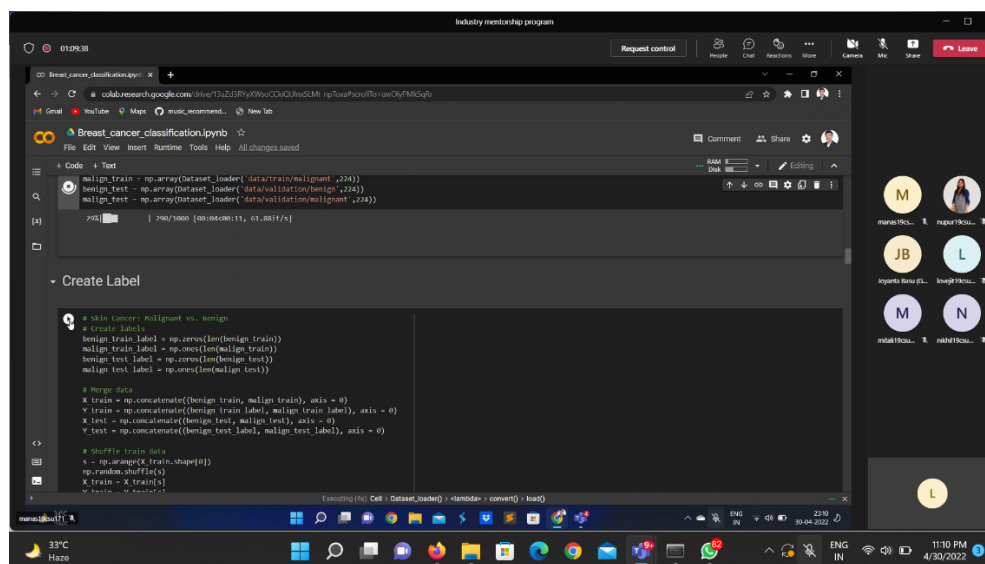
13 Annexure I: Responsibility Chart



Task	Nancy	Mitali	Lakshita	Madhur
Getting Started	R A C I	R A C I	R A C I	R A C I
Data Collection	R A C I	R A C I	R A C I	R A C I
Data preprocessing	R A C I	R A C I	R A C I	R A C I
Literature Review	R A C I	R A C I	R A C I	R A C I
Learning Best Models	R A C I	R A C I	R A C I	R A C I
Risk Identification	R A C I	R A C I	R A C I	R A C I
Training Deep Neural ...	R A C I	R A C I	R A C I	R A C I

Task	Nancy	Mitali	Lakshita	Madhur
Data preprocessing	R A C I	R A C I	R A C I	R A C I
Literature Review	R A C I	R A C I	R A C I	R A C I
Learning Best Models	R A C I	R A C I	R A C I	R A C I
Risk Identification	R A C I	R A C I	R A C I	R A C I
Training Deep Neural ...	R A C I	R A C I	R A C I	R A C I
Iterative Training	R A C I	R A C I	R A C I	R A C I
Deployment with Gradio	R A C I	R A C I	R A C I	R A C I

14 Annexure II: Screenshots of all the MS-Team meetings (online)/ handwritten comments(offline) from guide



Industry mentorship program

01:10:19

Request control

People Chat Reactions More Camera Mic Share Leave

Breast_cancer_classification.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Code Text

```
s = np.arange(X_test.shape[0])
np.random.shuffle(s)
X_test = X_test[s]
y_test = y_test[s]

# to categorical
y_train = to_categorical(y_train, num_classes=2)
y_test = to_categorical(y_test, num_classes=2)
```

Train and Evaluation split

```
[ ] x_train, x_val, y_train, y_val = train_test_split(
    X_train, y_train,
    test_size=0.2,
    random_state=11
)
```

Display Some Images

```
# display first 15 images of males, and how they are classified
w=10
h=10
fig=plt.figure(figsize=(15, 15))
columns = 4
```

completed at 11:11 PM

33°C Haze

ENG IN 23:11 30-04-2022

11:11 PM 4/30/2022

Participants: manan19cou171, nupur19cou..., JB, L, Hyarta Basi G..., kimg19cou..., M, N, mital19cou..., mital19cou...

Industry mentorship program

01:10:24

Request control

People Chat Reactions More Camera Mic Share Leave

Breast_cancer_classification.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Code Text

```
ax = fig.add_subplot(rows, columns, 1)
if np.argmax(y_train[i]) == 0:
    ax.title.set_text('benign')
else:
    ax.title.set_text('malignant')
plt.imshow(x_train[i], interpolation='nearest')
plt.show()
```

Executing (0s)

33°C Haze

ENG IN 23:11 30-04-2022

11:11 PM 4/30/2022

Participants: manan19cou171, nupur19cou..., JB, L, Hyarta Basi G..., kimg19cou..., M, N, mital19cou..., mital19cou...

Industry mentorship program

01:10:30

Request control

People Chat Reactions More Camera Mic Share Leave

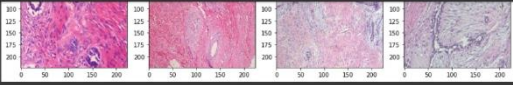
colab.research.google.com/8hw/13a2/d30tyWwvC3dOJlhwSdMI-npIowahcmfTo-CvWwQX5gGj

Breast_cancer_classification.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[0] [4]



Data Generator

```
BATCH_SIZE = 16

# using original generator
train_generator = ImageDataGenerator(
    zoom_range=2, # set range for random zoom
    rotation_range = 90,
    horizontal_flip=True, # randomly flip images
    vertical_flip=True, # randomly flip images
)
```

Model: ResNet50

```
def build_model(backbone, lr=1e-4):
    model = Sequential()
    model.add(backbone)
```

Waiting for colab.research.google.com...

33°C Haze

11:11 PM 4/30/2022

Participants:

- M manv19c...
- JB JB
- Kyanda Bana (S... kony19c...
- M M
- N N
- mitah19c...
- mitah19c...
- L

15 Annexure III: Sample of dataset

The image displays a Google Drive interface with a sidebar on the left containing navigation options: New, Priority, My Drive, Computers, Shared with me, Recent, Starred, Trash, and Storage (12.4 GB used). The main content area shows the 'skin_burn' folder, which contains two subfolders: 'Test' and 'Train'. The 'Train' folder is selected, showing three subfolders: 'degree0', 'degree1', and 'degree2'. The 'degree0' folder is selected, displaying a grid of 12 skin burn images. The images are labeled as follows:

- Row 1: img0.jpg, img3.jpg, img4.jpg, img10.jpg
- Row 2: img13.jpg, img14.jpg, img16.jpg, img18.jpg
- Row 3: img1.jpg, img2.jpg, img5.jpg, img7.jpg
- Row 4: img8.jpg, img9.jpg, img11.jpg, img12.jpg

The images show various stages and types of skin burns, including first-degree burns, second-degree blisters, and third-degree burns. The interface also includes a search bar at the top and a right sidebar with additional navigation options.

New

✓ Priority

▶ My Drive

▶ Computers

Shared with me

Recent

☆ Starred

Trash

Storage

12.4 GB used

My Drive > skin_burn > Train > degree1 ▾

☰

ⓘ

📁

📁

✓

+

>



img249.jpg



img325.jpg



img326.jpg



img333.jpg



img334.jpg



img335.jpg



img349.jpg



img350.jpg