DETECTION AND CLASSIFICATION OF BRAIN TUMOR USING DEEP LEARNING

CSE 4019 IMAGE PROCESSING

Report submitted in partial fulfilment of the requirements for the degree of

Bachelor of Technology

by

Sreejan Chattopadhyay (19BCE1541)

DEVANG PATHAK (19BCE1559)

ISHITA KUMAR (19BCE1551)

under the guidance of

DR. GEETHA BALAN

SCOPE

VIT®

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

MAY 2022

BONAFIDE CERTIFICATE

This is to certify that this project entitled "Detection and Classification of Brain tumor using deep learning" submitted in partial fulfilment of the

degree of Bachelor of Technology to Vellore Institute of Technology, Chennai, done by Devang **Pathak, Ishita Kumar, Sreejan Chattopadhyay** Register No**. 19BCE1559, 19BCE1551, 19BCE1541** are an authentic work carried out by under my guidance. The matter embodied in this project work has not been submitted earlier for award of any degree or diploma to the best of my knowledge and belief.

Place: VIT Chennai
Date: 12-05-22

Signature of the Course Faculty

The report is satisfactory / unsatisfactory

Signature of the Internal Examiner **Signature of the External Examiner**

Approved by
Head of the Department

DECLARATION

We **Devang Pathak, Ishita Kumar, Sreejan Chattopadhyay** bearing Reg. No. **19BCE1559, 19BCE1551, 19BCE1541** respectively hereby declare that this project report entitled **Detection and Classification of Brain tumor using deep learning** has been prepared towards the partial fulfilment of the requirement of the course Image Processing under the guidance of **DR. GEETHA BALAN**

I also declare that this project report is my original work and has not been previously submitted for the award of any Degree, Diploma, Fellowship, or other similar titles.

ABSTRACT

Classification of brain tumors is one of the most important and time-consuming tasks in the field of medical imaging because manual classification by humans can lead to inaccurate predictions and diagnoses. What's more, days exacerbating than
task when you have a lot of data to help. Because brain tumors vary widely in shape and similarities exist between tumors and normal tissues, it is irresistible to extract tumor regions from images.
In this project, we have conducted a comparative study of 3 popular machine learning models that are InceptionNetV3, MobileNetV2 and Convolution Neural network.
Keywords: InceptionNetV3, MobileNetV2, CNN, Medical Imaging

ACKNOWLEDGEMNT

We are using this opportunity to express our gratitude to everyone who supported us throughout the course of this project. We are thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. We are sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

We wish to place on record the profound gratitude to our guide, Dr. Geetha Balan, SCOPE, Vellore Institute of Technology, Chennai for her constant encouragement, continued support and valuable guidance without which this work could not have been possible. The support and encouragement given by our friends and colleagues are beyond words and we are deeply indebted to her for her continued help and dedication.

TABLE OF CONTENTS

INTRODUCTION

0. MEDICAL IMAGING

Medical imaging, also known as radiology, is the field of medicine in which medical professionals recreate various images of parts of the body for diagnostic or treatment purposes. Medical imaging procedures include noninvasive tests that allow doctors to diagnose injuries and diseases without being intrusive.

Medical imaging is a central part of the improved outcomes of modern medicine. Different types of medical imaging procedures include:

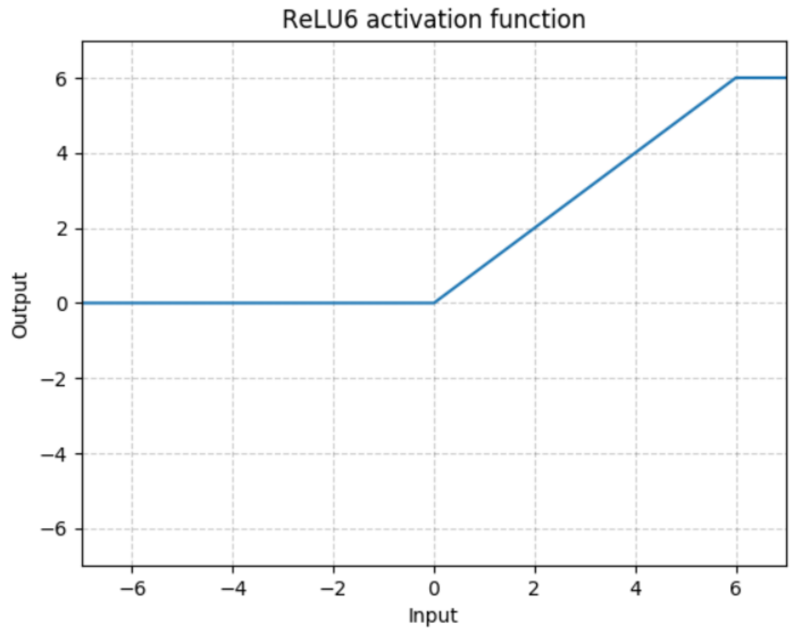
- Xrays
- Magnetic resonance imaging (MRI)
- Ultrasounds
- Endoscopy
- Tactile imaging
- Computerized tomography (CT scan)

Other beneficial medical imaging procedures include nuclear medicine functional imaging techniques, such as positron emission tomography (PET) scans. Other uses of medical imaging include scans to see how well your body is responding to a treatment for a fracture or illness.

1. INTRODUCTION TO MobileNetV2

MobileNet was first developed as a model using depthwise seperable convolution which drastically reduced the model size and computation cost which is better for mobile devices with low computation power.
MobileNetV1 has two layers.
The first layer is called a depth convolution. Perform simplified filtering by applying one convolution filter per input channel.

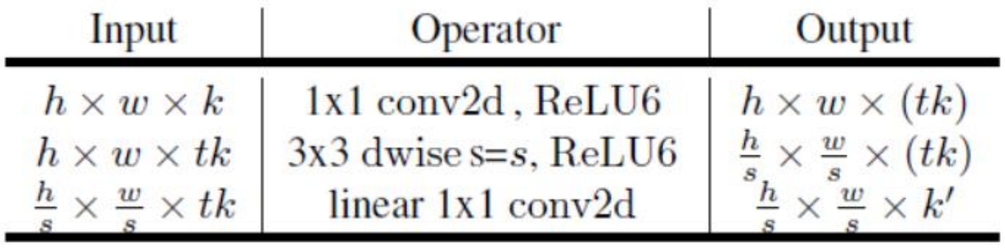
The second layer is a 1x1 convolution called point convolution, which is responsible for computing linear combinations of input channels to build new features.
ReLU6 is used here for comparison. (Actually I can't find a hint in the MobileNetV1 tech report that they're using ReLU6… You might need to check the code on Github…), i.e. min(max(x, 0), 6) like this:



ReLU6

But after this a second version of MobileNet was developed, that came to be known as MobileNet V2
There are two types of blocks in MobileNetV2. One is a residual block with a stride length of 1, and the other is a block with a stride length of 2 to reduce the

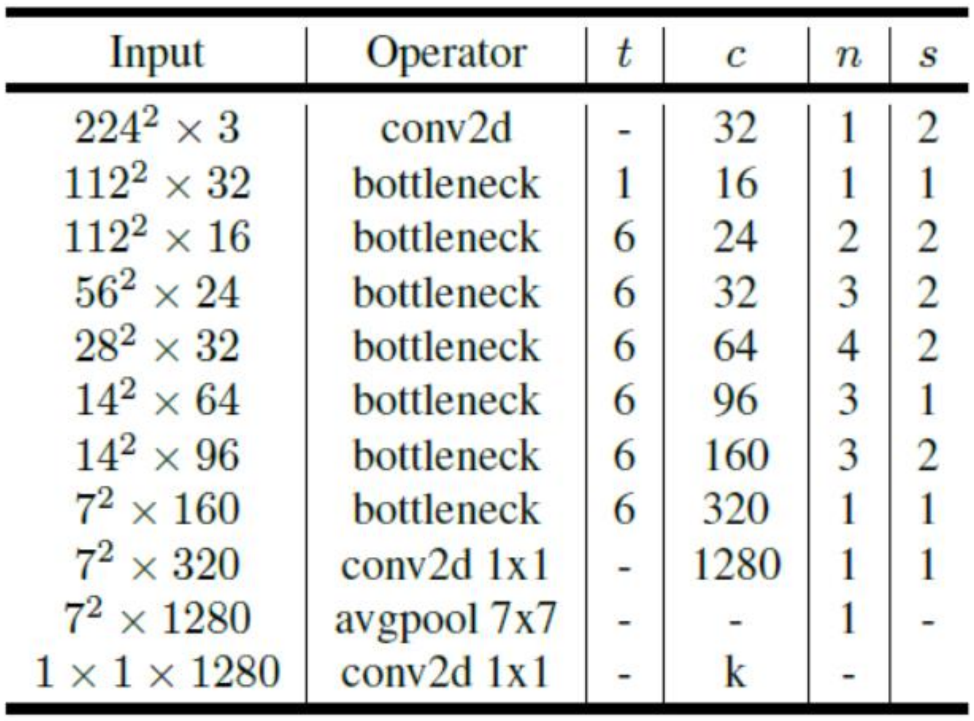
size.There are 3 layers for both block types. This time the first layer is a 1x1 convolution with ReLU6. The second layer is a depth convolution.
3rd layer is another 1x1 convolution but no non-linearity. It is argued that if ReLU is used again, the deep network will only have the power of the linear classifier in the non-zero volume fraction of the output domain.



Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Figure 1. Operators and Outputs

1.1. OVERALL ARCHITECTURE CONFIGURATION OF MOBILENET V2



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 2. Overall Architecture

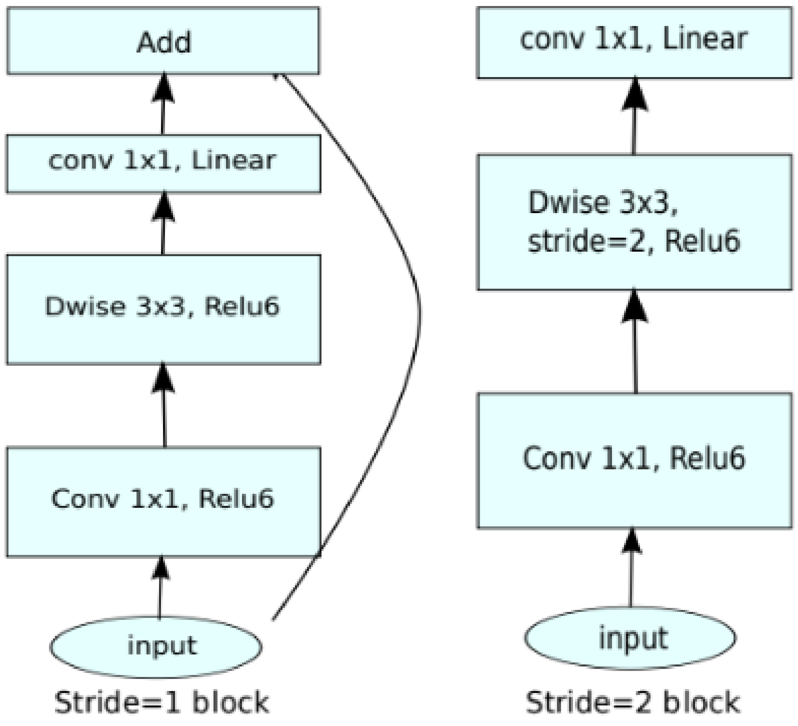


Figure 3. Flow of overall architecture

where *t*: expansion factor, *c*: number of output channels, *n*: repeating number, s: stride. 3×3 kernels are used for spatial convolution. In typical, the primary network (width multiplier 1, **224×224**), has a computational cost of 300 million multiply-adds and uses **3.4 million parameters**. The performance trade offs are further explored, for **input resolutions from 96 to 224**, and **width multipliers of 0.35 to 1.4**. The network computational cost up to 585M MAdds, while the model size vary between 1.7M and 6.9M parameters. To train the network, 16 GPU is used with batch size of 96.

2. INTRODUCTION TO InceptionNetV3 Model

For understanding the InceptionNet V3 Model we need to understand how it evolved from its previous versions of V1 and V2.

2.1 Where it all started- InceptionNet V1

The problem:
Salient parts in the image can have extremely large variation in size. For instance, an image with a dog can be either of the following, as shown below. The area occupied by the dog is different in each image.
Because of this huge variation in the location of the information, choosing the right kernel size for the convolution operation becomes tough. A larger kernel is preferred for information that is distributed more globally, and a smaller kernel is preferred for information that is distributed more locally.
Very deep networks are prone to overfitting. Broadcasting gradient updates over the network is also difficult. The naive stacking of large convolutions is computationally expensive.

The Solution:
Why not use filters of different sizes at the same level? Essentially, the network will be a little "wider" rather than "deep". The authors designed the initial module to reflect the same.
The image below shows the "naive" initial module. Convolution on the input using filters of three sizes (1x1, 3x3, 5x5). Also, maximum union is performed. The outputs are combined and sent to the next startup module.

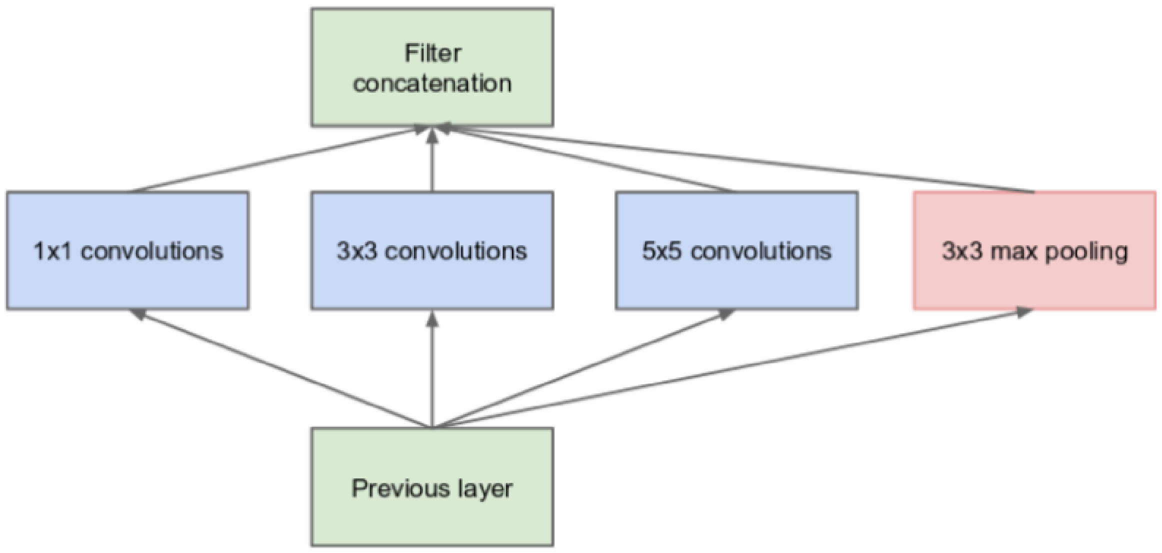


Figure 4. Inception Module, Naive version

2.2. Inception V2

Inception v2 and Inception v3 were presented in the same paper. The authors proposed a number of upgrades which increased the accuracy and reduced the computational complexity. Inception v2 explores the following:

The Premise:
Reduce representational bottleneck. The intuition was that, neural networks perform better when convolutions didn't alter the dimensions of the input drastically. Reducing the dimensions too much may cause loss of information, known as a "representational bottleneck"
Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.

The Solution:

Speed up the computation by decomposing the
5x5 convolution into two 3x3 convolutions. It may seem counterintuitive, but a 5x5 convolution is 2.78x more expensive than a 3x3 convolution.
So concatenating two 3x3 convolutions will actually give you
better performance. This is shown in the image below.

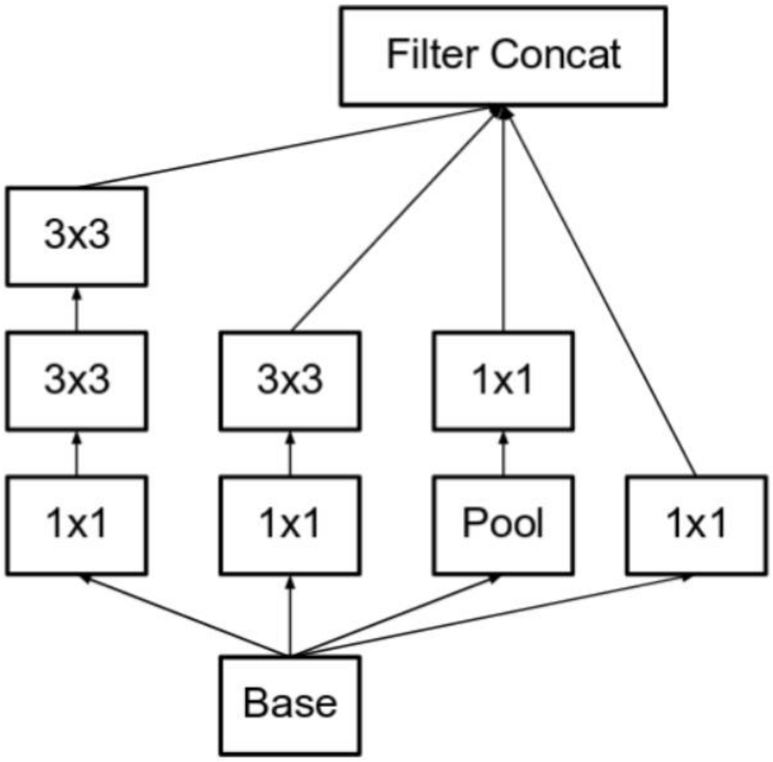


Figure 5. The left-most 5x5 convolution of the old inception module, is now represented as two 3x3 convolutions.

Moreover, they factorize convolutions of filter size nxn to a combination of 1xn and nx1 convolutions. For example, a 3x3 convolution is equivalent to first performing a 1x3 convolution, and then performing a 3x1 convolution on its output. They found this method to be 33% more cheaper than the single 3x3 convolution. This is illustrated in the below image.

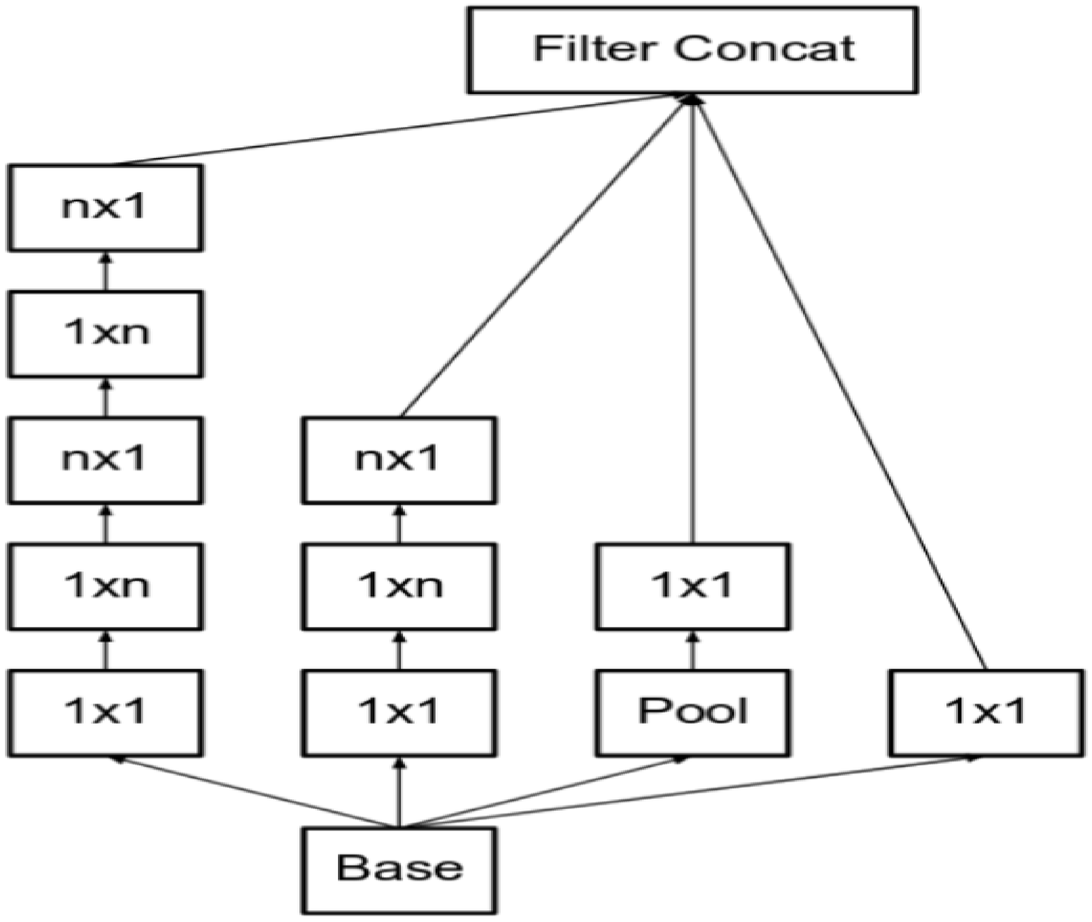
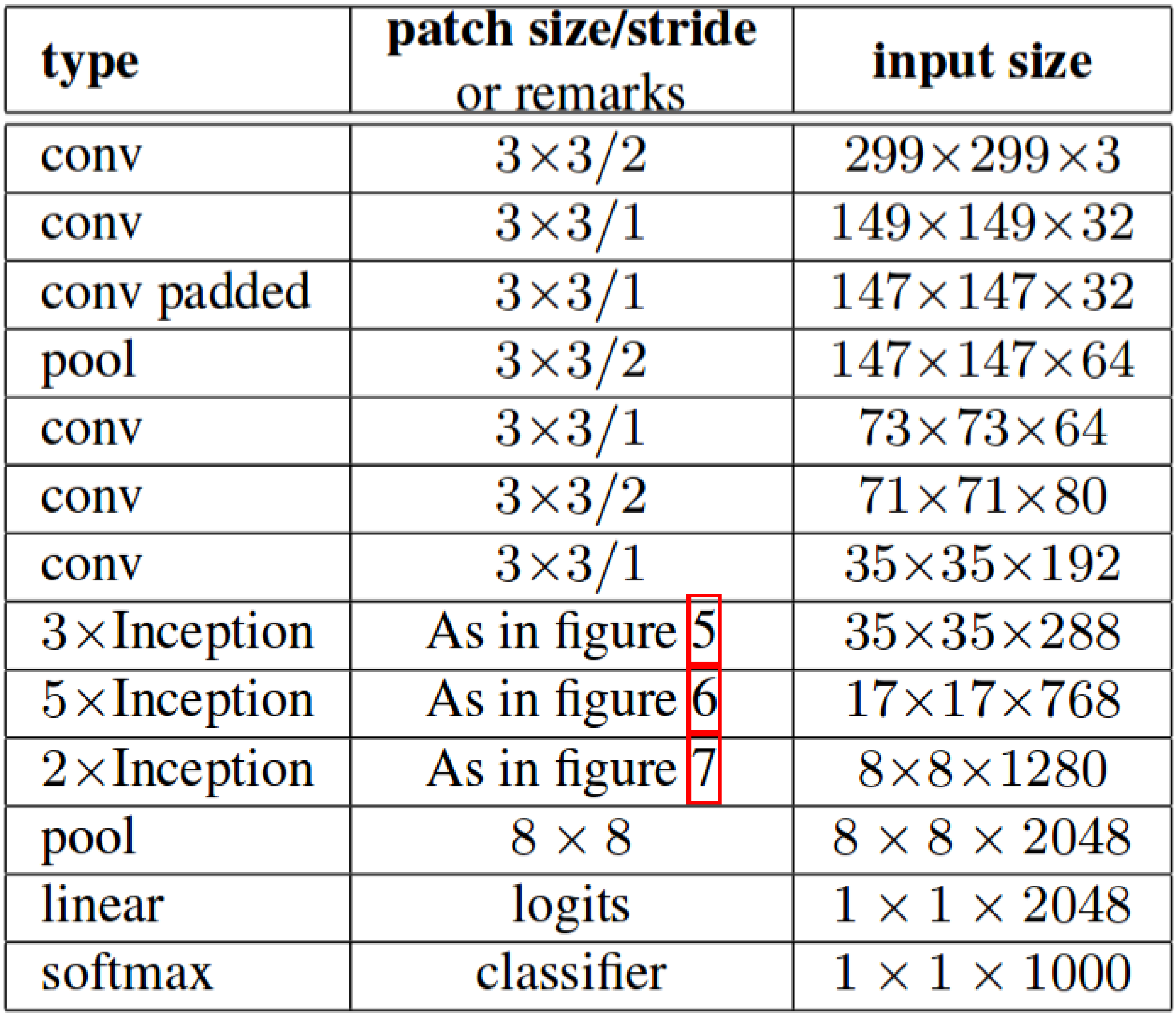


Figure 6. Here, put n=3 to obtain the equivalent of the previous image. The left-most 5x5 convolution can be represented as two 3x3 convolutions, which inturn are represented as 1x3 and 3x1 in series.

The above three principles were used to build three different types of inception modules (Let's call them modules **A, B** and **C** in the order they were introduced. These names are introduced for clarity, and not the official names). The architecture is as follows:



type	patch size/stride or remarks	input size
conv	$3\times3/2$	$299\times299\times3$
conv	$3\times3/1$	$149\times149\times32$
conv padded	$3\times3/1$	$147\times147\times32$
pool	$3\times3/2$	$147\times147\times64$
conv	$3\times3/1$	$73\times73\times64$
conv	$3\times3/2$	$71\times71\times80$
conv	$3\times3/1$	$35\times35\times192$
$3\times$Inception	As in figure 5	$35\times35\times288$
$5\times$Inception	As in figure 6	$17\times17\times768$
$2\times$Inception	As in figure 7	$8\times8\times1280$
pool	8×8	$8\times8\times2048$
linear	logits	$1\times1\times2048$
softmax	classifier	$1\times1\times1000$

Figure 7. OVERALL ARCHITECTURE

2.3. **InceptionNet V3**

The premise:
The authors noted that secondary classifiers did
not contribute significantly until near the end of the training process when the
accuracy was approaching the limit. They claimed to act like a
normalizer, especially if you have a BatchNorm or Dropout operation. The

improvements in
Inception v2 should be explored without major module changes.

The Solution
Inception Net v3 contains all of the above updates released for
Inception v2 and additionally uses:
RMSProp Optimizer.
Factorized Convolution 7x7. BatchNorm of the
auxiliary classifier.
Label smoothing (a kind of regularization component added to the loss formula to prevent the network from becoming too class-confident; to avoid reassignment).

3. CONVOLUTIONAL NEURAL NETWORK

Over the past few decades, deep learning has proven to be a very powerful tool due to its ability to process large amounts of data. Interest in the use of hidden layers has surpassed traditional methods, especially in the field of pattern recognition. One of the most widely used deep neural networks is the convolutional neural network.

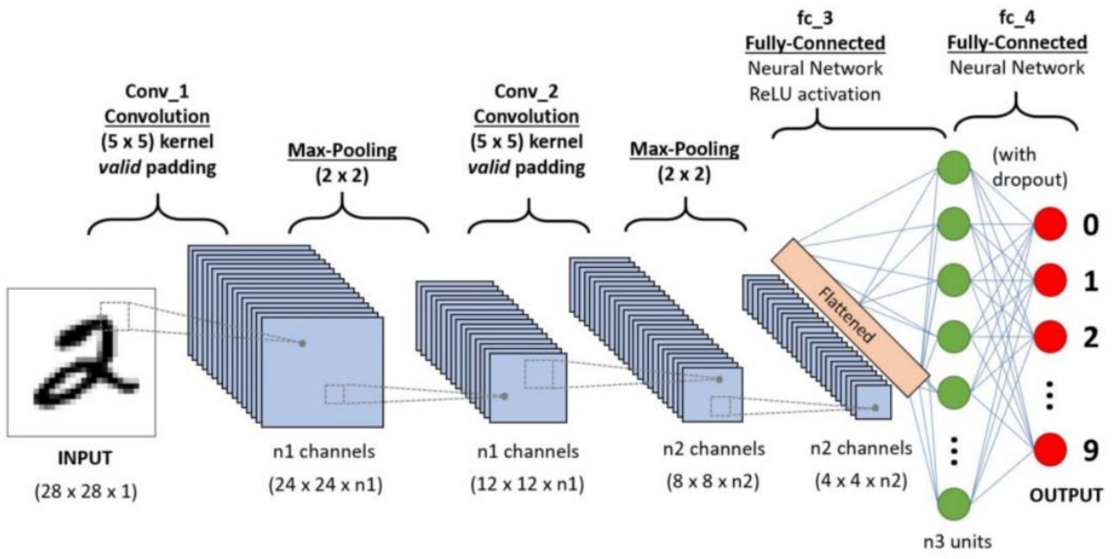


Figure 8. Overall Architecture flow

A convolutional neural network consists of several layers of artificial neurons. An artificial neuron, a crude imitation of its biological counterpart, is a mathematical function that computes the weighted sum of multiple inputs and outputs an activation value. When you feed an image into a ConvNet, each layer generates multiple activation functions that are passed to the next layer.

The first layer usually extracts key features like horizontal or diagonal edges. This output is passed to the next level of detecting more complex objects such as corners or joined edges. The deeper you go into the network, the more complex features like objects, faces, etc. can be identified.

Based on the activation map of the final convolution layer, the classification layer outputs a set of confidence scores (values between 0 and 1) that specify how likely the image is to belong to a "class." For instance, if you have a ConvNet that detects cats, dogs, and horses, the output of the final layer is the possibility that the input image contains any of those animals.

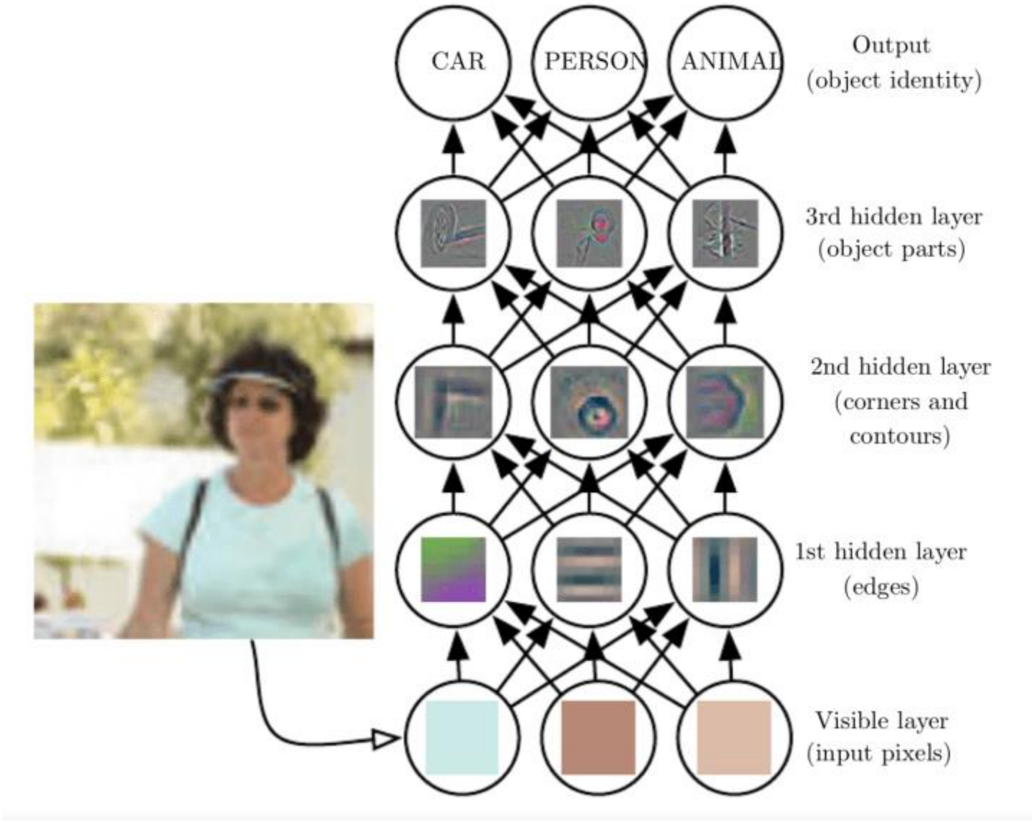


Figure 9. Various hidden layers in CNN

REVIEW OF LITERATURE

Paper1: Rethinking the Inception Architecture for Computer Vision

Convolutional networks are at the core of most stateoftheart computer vision solutions for a wide variety of tasks. Since 2014 very deep convolutional networks started to become mainstream, yielding substantial gains in various benchmarks. Although increased model size and computational cost tend to translate to immediate quality gains for most tasks (as long as enough labeled data is provided for training), computational efficiency and low parameter count are still enabling factors for various use cases such as mobile vision and bigdata scenarios. Here we explore ways to scale up networks in ways that aim at utilizing the added computation as efficiently as possible by suitably factorized convolutions and aggressive regularization. We benchmark our methods on the ILSVRC 2012 classification challenge validation set demonstrate substantial gains over the state of the art: 21.2% top1 and 5.6% top5 error for single frame evaluation using a network with a computational cost of 5 billion multiplyadds per inference and with using less than 25 million parameters. With ensemble of **four** models and **multi-crop scoring,** we report **a top 5 error of** 3.5% **in** the validation set (3.6% error **in** the test set) and **a top 1 error of** 17.3% **in** the validation set.

Paper2: MobileNetV2: Inverted Residuals and Linear Bottlenecks

In this paper authors describe a new mobile architecture, MobileNetV2, that improves the state of the art performance of mobile models on multiple tasks and benchmarks as well as across a spectrum of different model sizes. We also describe efficient ways of applying these mobile models to object detection in a novel framework we call SSDLite. Additionally, we demonstrate how to build mobile

semantic segmentation models through a reduced form of DeepLabv3 which we call Mobile DeepLabv3.
The MobileNetV2 architecture is based on an inverted residual structure where the input and output of the residual block are thin bottleneck layers opposite to traditional residual models which use expanded representations in the input an MobileNetV2 uses lightweight depthwise convolutions to filter features in the intermediate expansion layer. Additionally, we find that it is important to remove non-linearities in the narrow layers in order to maintain representational power. We demonstrate that this improves performance and provide an intuition that led to this design. Finally, our approach allows decoupling of the input/output domains from the expressiveness of the transformation, which provides a convenient framework for further analysis. We measure our performance on Imagenet classification, COCO object detection, VOC image segmentation. We evaluate the trade-offs between accuracy, and number of operations measured by multiply-adds (MAdd), as well as the number of parameters

Paper3: MixConv: Mixed Depthwise Convolutional Kernels : Depthwise convolution is becoming increasingly popular in modern efficient ConvNets, but its kernel size is often overlooked. In this paper, we systematically study the impact of different kernel sizes, and observe that combining the benefits of multiple kernel sizes can lead to better accuracy and efficiency. Based on this observation, we propose a new mixed depthwise convolution (MixConv), which naturally mixes up multiple kernel sizes in a single convolution. As a simple drop-in replacement of vanilla depthwise convolution, our MixConv improves the accuracy and efficiency for existing MobileNets on both ImageNet classification and COCO object detection.

Paper4: Fast Sparse CovNet

Historically, the pursuit of efficient inference has been one of the driving forces behind research into new deep learning architectures and building blocks. Some recent examples include: the squeezeandexcitation module, depthwise separable convolutions in Xception, and the inverted bottleneck in MobileNet v2. Notably, in all of these cases, the resulting building blocks enabled not only higher efficiency, but also higher accuracy, and found wide adoption in the field. In this work, we further expand the arsenal of efficient building blocks for neural network architectures; but instead of combining standard primitives (such as convolution), we advocate for the replacement of these dense primitives with their sparse counterparts. While the idea of using sparsity to decrease the parameter count is not new, the conventional wisdom is that this reduction in theoretical FLOPs does not translate into realworld efficiency gains.

PROJECT SYSTEM CONFIGURATION

1. PROCESSOR: INTEL CORE I7
2. SYSTEM MEMORY: 16GB
3. NVIDIA GEFORCE MX250 3GB

SUGGESTION: If processing on system taking too much time then can choose platforms like Google COLAB.

PROJECT FLOW CHART

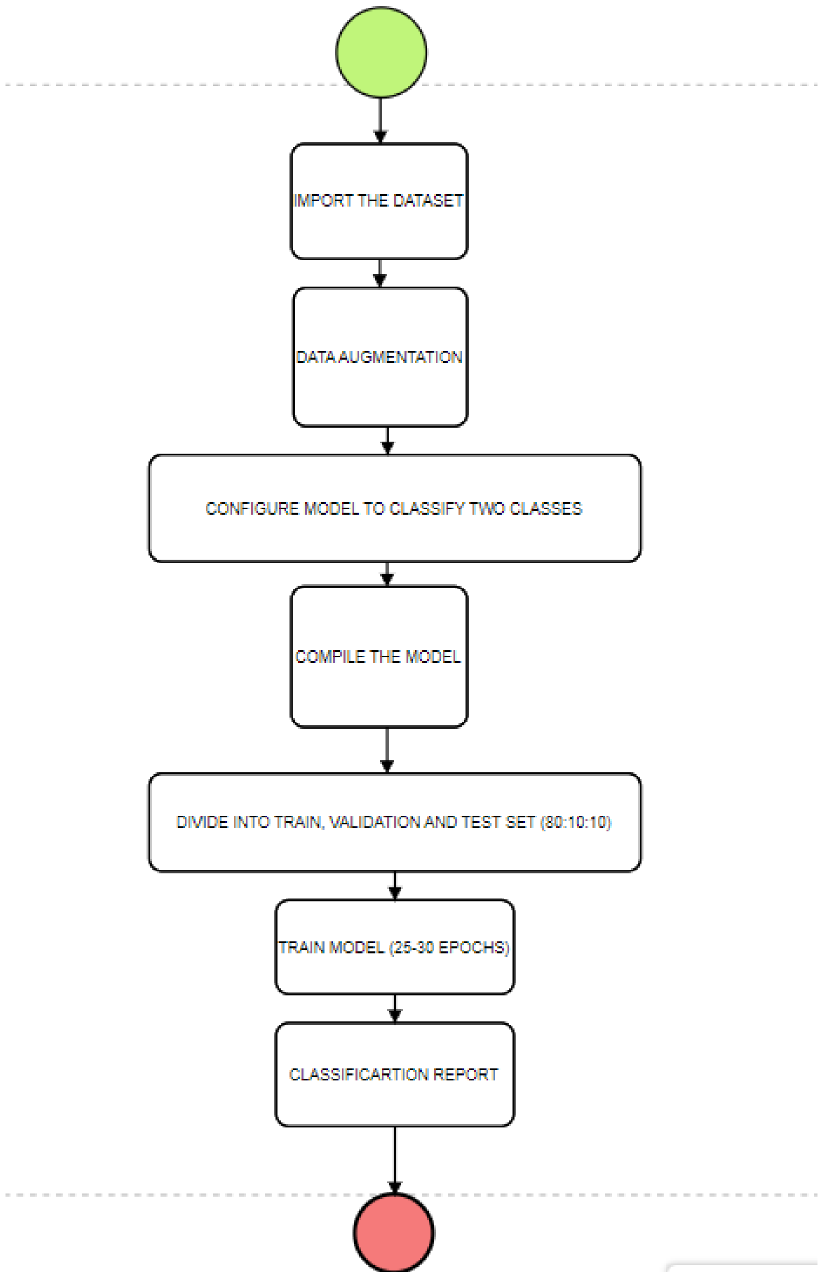

IMPORT THE DATASET
DATA AUGMENTATION
CONFIGURE MODEL TO CLASSIFY TWO CLASSES
COMPILE THE MODEL
DIVIDE INTO TRAIN, VALIDATION AND TEST SET (80:10:10)
TRAIN MODEL (25-30 EPOCHS)
CLASSIFICARTION REPORT

Figure 10. Generic flow chart for the project

DATASET DESCRIPTION

The dataset used has been sourced from Kaggle. It contains two types of brain MRI images. One which have been detected with brain tumor. Other which doesn’t have brain tumor. The dataset contains a total of 253 images.

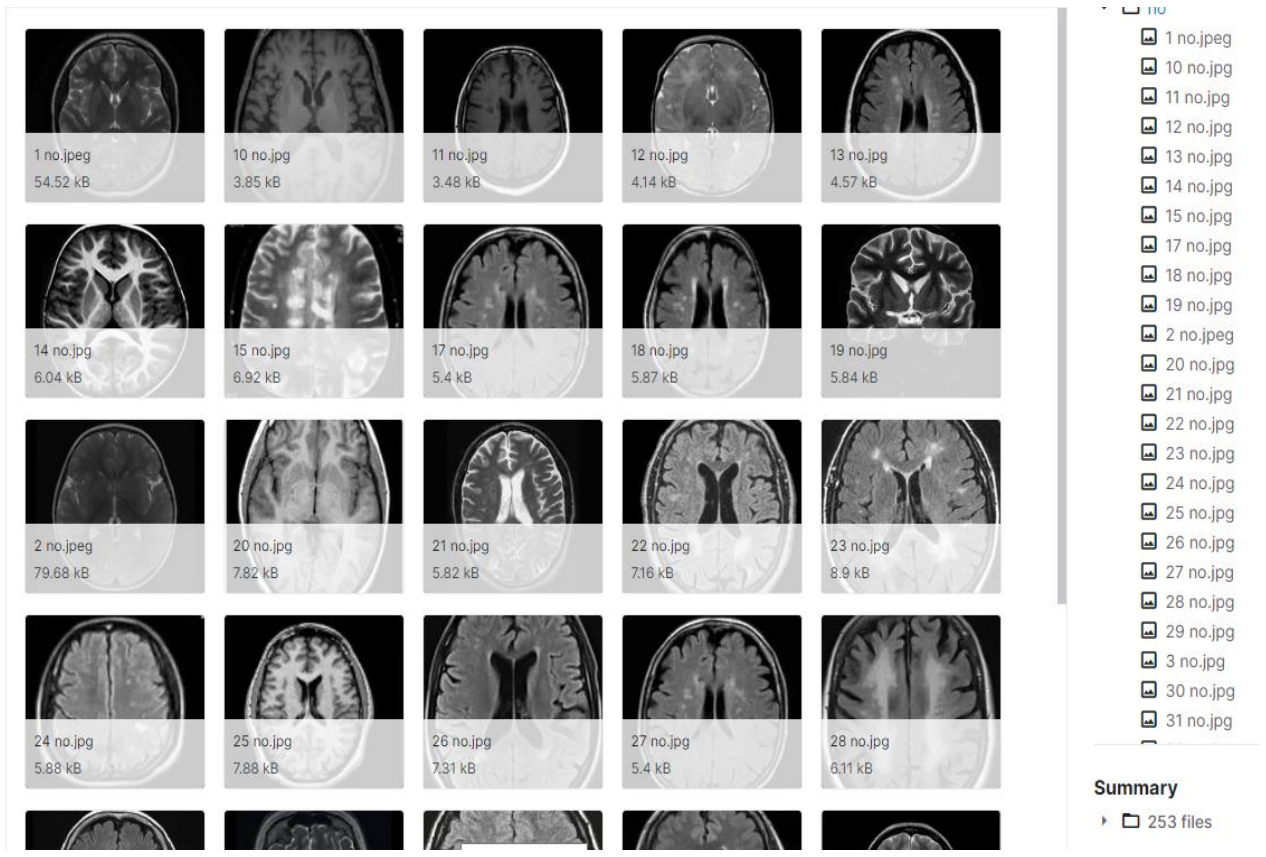


Figure 11. Dataset of images

DATA AUGMENATION AND PREPROCESSING

The dataset we found most suitable was a public dataset available on Kaggle- BRAIN MRI IMAGES FOR BRAIN TUMOR DETECTION

About the dataset: The dataset contains two separate files/collection of data. First file contains 155 Brain MRI Images that are tumorous and the other collection contains 98 Brain MRI images that are non-tumorous. Image put into the model was of dimension 224 x 224

Challenge: The biggest challenge was that running a neural network needs large datasets for better results but our dataset is limited. And further we will divide it in 80% for training and leftover for testing and validation purposes.

So, as described in the previous slide. We needed larger dataset. This calls for data augmentation to format images and create different versions of similar content in order to expose the model to a wider range array of training examples. We applied the following for data requirements:

1. Rotation with range 10.

2. Width shift with range 0.1

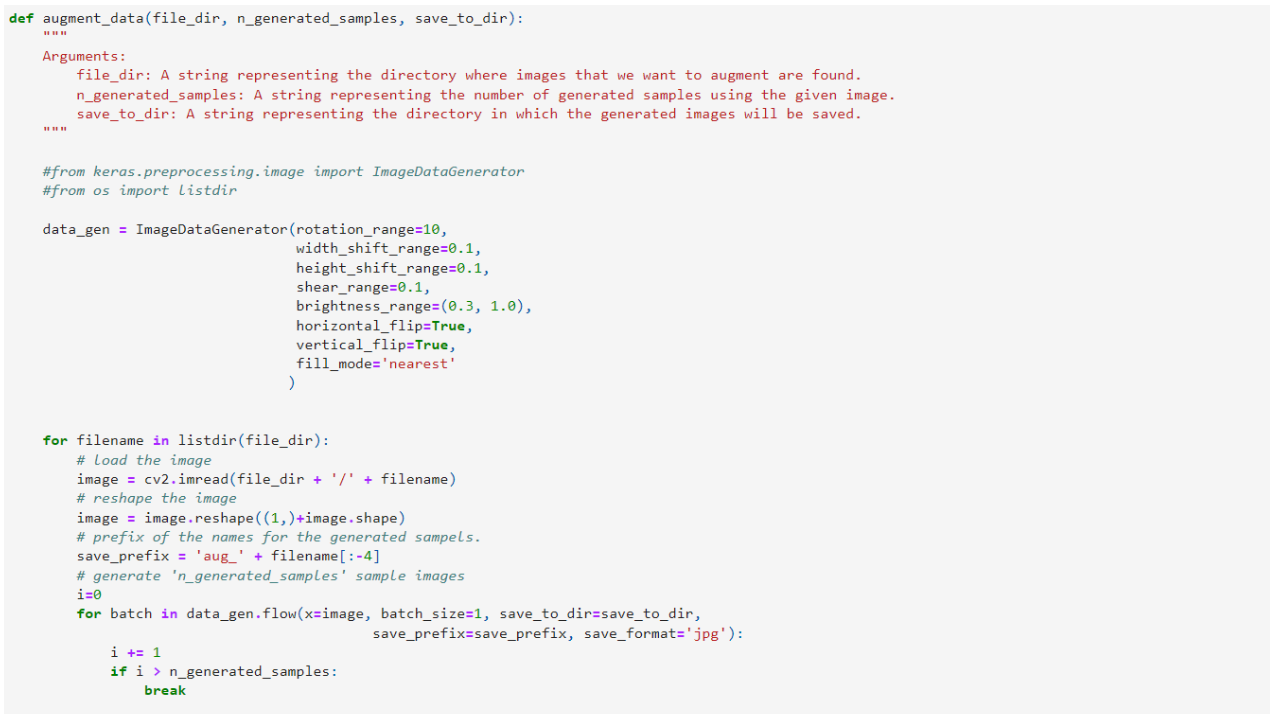
3. Shear with range 0.1

4. Brightness was set with range 0.3 to 1.0

5. Horizontal flip and vertical flip was also performed.

6. Fill mode set to nearest.

CODE SNIPPET

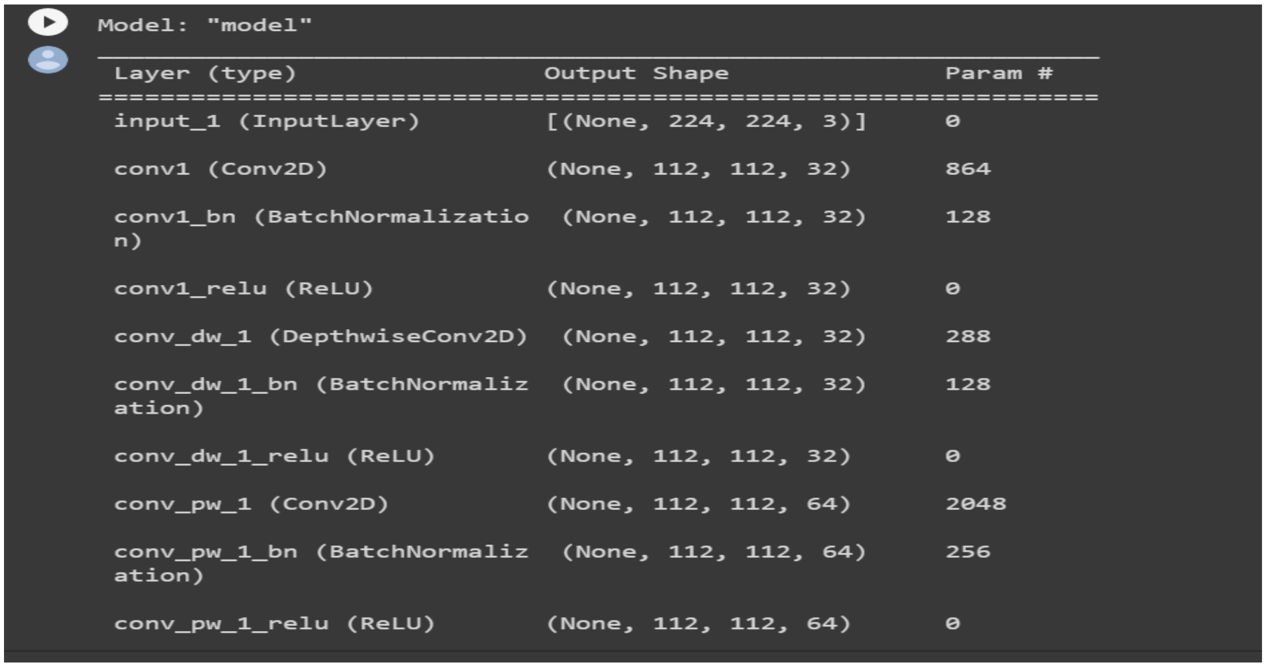


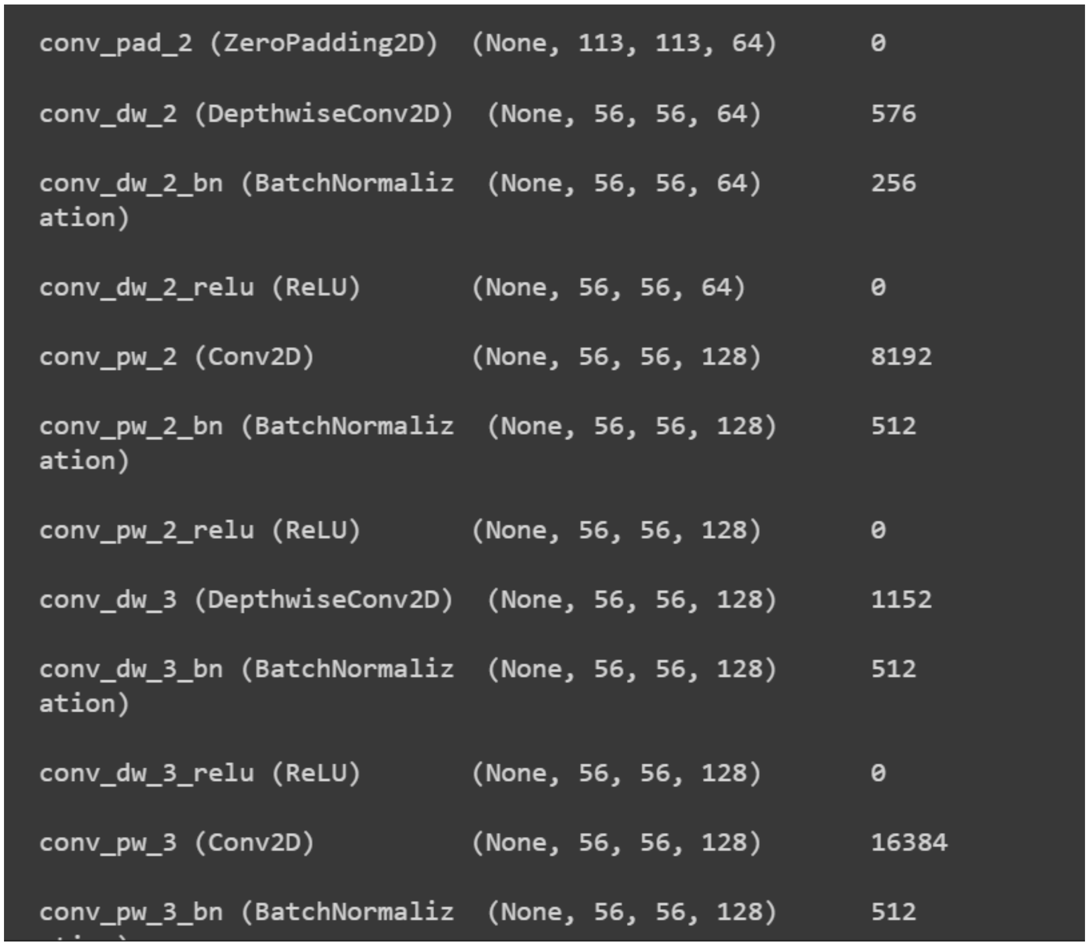
```python
def augment_data(file_dir, n_generated_samples, save_to_dir):
    """
    Arguments:
        file_dir: A string representing the directory where images that we want to augment are found.
        n_generated_samples: A string representing the number of generated samples using the given image.
        save_to_dir: A string representing the directory in which the generated images will be saved.
    """

    #from keras.preprocessing.image import ImageDataGenerator
    #from os import listdir

    data_gen = ImageDataGenerator(rotation_range=10,
                                  width_shift_range=0.1,
                                  height_shift_range=0.1,
                                  shear_range=0.1,
                                  brightness_range=(0.3, 1.0),
                                  horizontal_flip=True,
                                  vertical_flip=True,
                                  fill_mode='nearest'
                                 )


    for filename in listdir(file_dir):
        # load the image
        image = cv2.imread(file_dir + '/' + filename)
        # reshape the image
        image = image.reshape((1,)+image.shape)
        # prefix of the names for the generated sampels.
        save_prefix = 'aug_' + filename[:-4]
        # generate 'n_generated_samples' sample images
        i=0
        for batch in data_gen.flow(x=image, batch_size=1, save_to_dir=save_to_dir,
                                   save_prefix=save_prefix, save_format='jpg'):
            i += 1
            if i > n_generated_samples:
                break
```

IMPLEMENTATION OF MOBILE NETV2

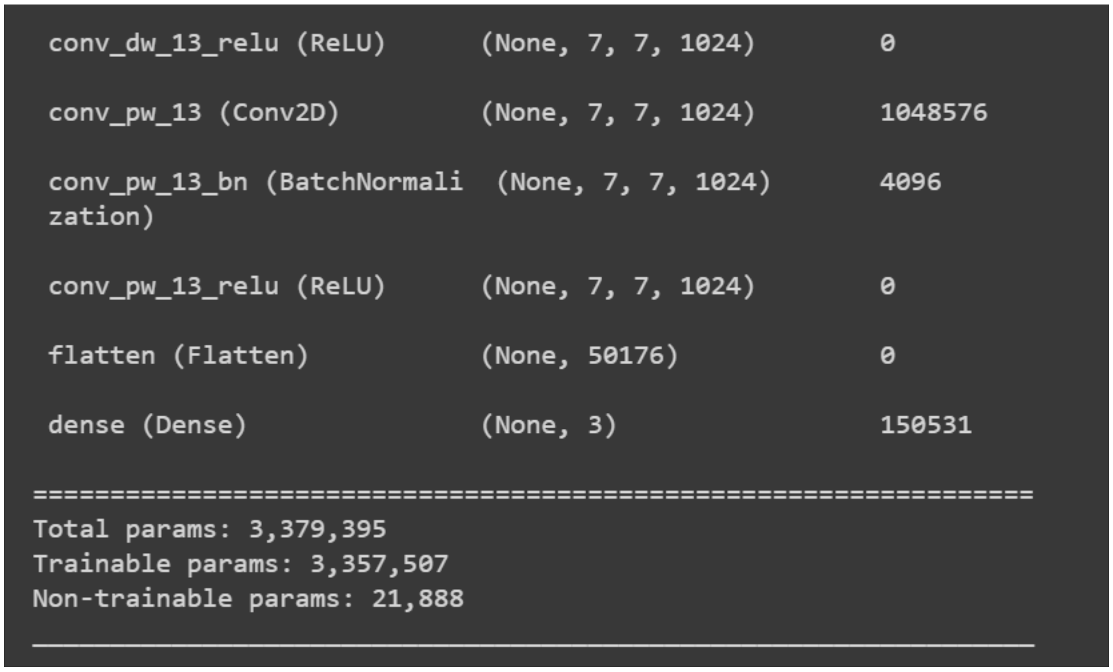
The first model that we trained is MobileNetV2 model. It is a predefined deep learning model from tensorflow. At first all the important libraries are being imported. After importing the libraries we import the data into google colab from google drive. The total count of images from the dataset were found out to be around 300 which is very less. Therefore, we first augment the dataset and increase the count of images trom 300 to 3000 images. After augmnentation we create our model for training and testing purpose. First we download the MobileNetV2 architecture from the tensorflow library and then configure it to classify the data with 2 classes. After creation of the model we compile the model. After compilation of the model we divide the dataset into train, validation and test set in the ratio 80:10:10 respectively. After division of the dataset we fit the model for training. After training for 25-30 epochs we create the confusion matrix and classification report. Classification report tells us that MobileNet model has achieved an accuracy of 94%.

MobileNet model structure


```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

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

 conv1_bn (BatchNormalizatio  (None, 112, 112, 32)     128
 n)

 conv1_relu (ReLU)           (None, 112, 112, 32)      0

 conv_dw_1 (DepthwiseConv2D)  (None, 112, 112, 32)     288

 conv_dw_1_bn (BatchNormaliz  (None, 112, 112, 32)     128
 ation)

 conv_dw_1_relu (ReLU)       (None, 112, 112, 32)      0

 conv_pw_1 (Conv2D)          (None, 112, 112, 64)      2048

 conv_pw_1_bn (BatchNormaliz  (None, 112, 112, 64)     256
 ation)

 conv_pw_1_relu (ReLU)       (None, 112, 112, 64)      0
```

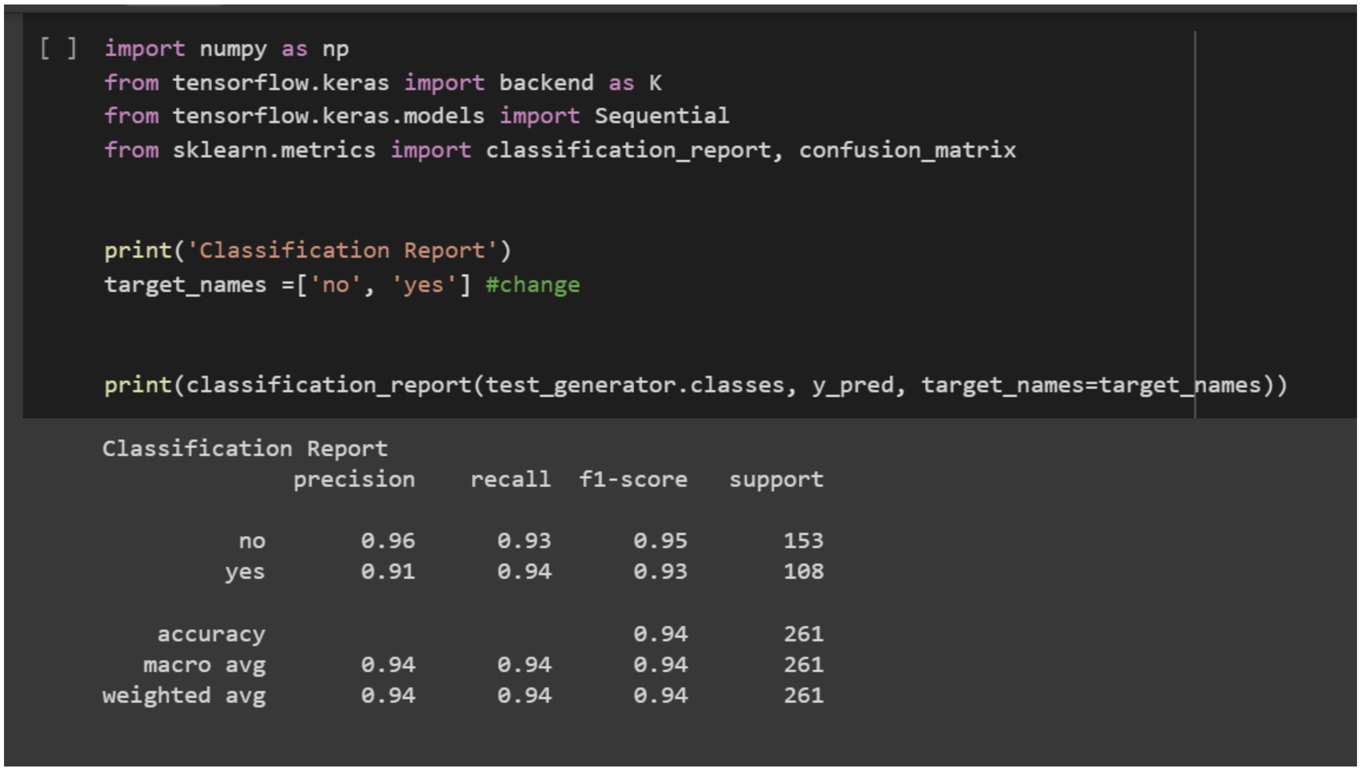


```
 conv_pad_2 (ZeroPadding2D)  (None, 113, 113, 64)      0

 conv_dw_2 (DepthwiseConv2D)  (None, 56, 56, 64)       576

 conv_dw_2_bn (BatchNormaliz  (None, 56, 56, 64)       256
 ation)

 conv_dw_2_relu (ReLU)       (None, 56, 56, 64)        0

 conv_pw_2 (Conv2D)          (None, 56, 56, 128)       8192

 conv_pw_2_bn (BatchNormaliz  (None, 56, 56, 128)      512
 ation)

 conv_pw_2_relu (ReLU)       (None, 56, 56, 128)       0

 conv_dw_3 (DepthwiseConv2D)  (None, 56, 56, 128)      1152

 conv_dw_3_bn (BatchNormaliz  (None, 56, 56, 128)      512
 ation)

 conv_dw_3_relu (ReLU)       (None, 56, 56, 128)       0

 conv_pw_3 (Conv2D)          (None, 56, 56, 128)       16384

 conv_pw_3_bn (BatchNormaliz  (None, 56, 56, 128)      512
```



```
 conv_dw_13_relu (ReLU)      (None, 7, 7, 1024)        0

 conv_pw_13 (Conv2D)         (None, 7, 7, 1024)        1048576

 conv_pw_13_bn (BatchNormali  (None, 7, 7, 1024)       4096
 zation)

 conv_pw_13_relu (ReLU)      (None, 7, 7, 1024)        0

 flatten (Flatten)           (None, 50176)             0

 dense (Dense)               (None, 3)                 150531

=================================================================
Total params: 3,379,395
Trainable params: 3,357,507
Non-trainable params: 21,888
_________________________________________________________________
```

Model Accuracy:



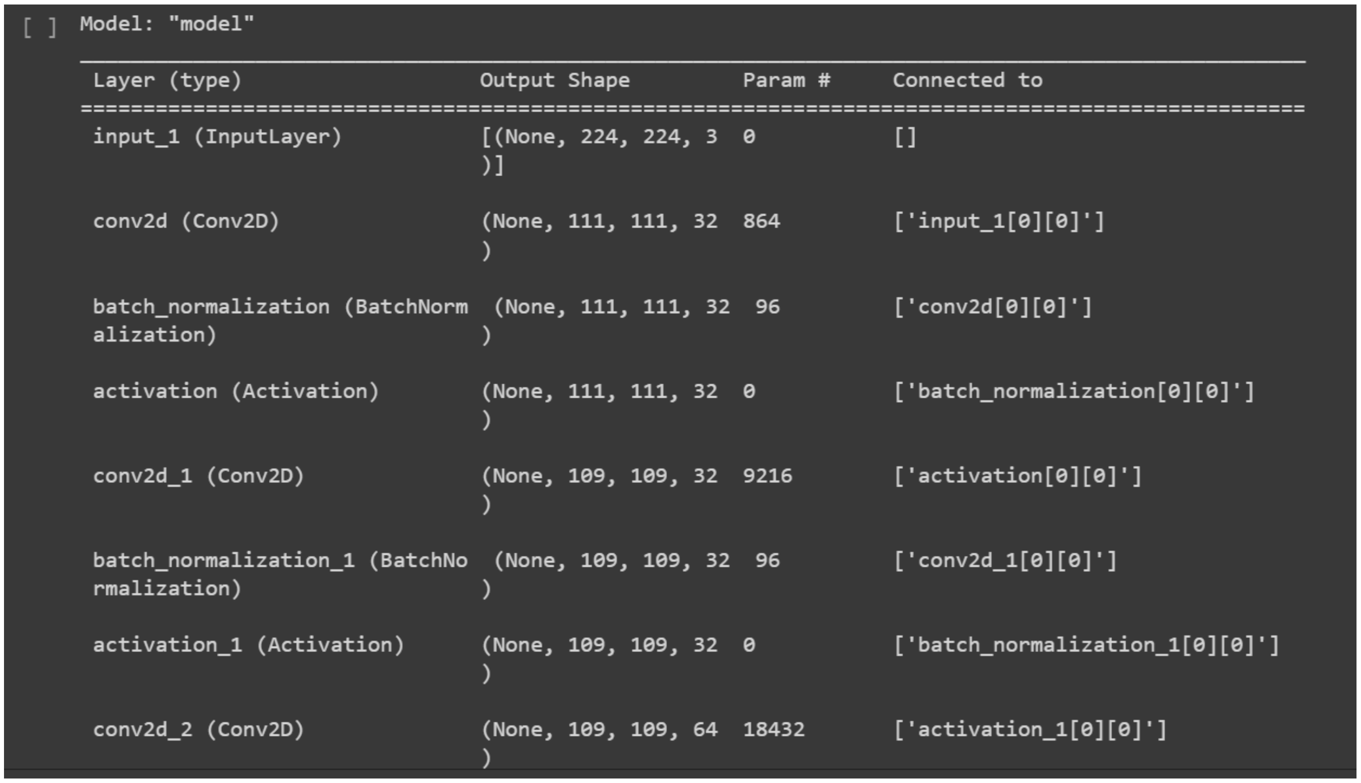
```
[ ]  import numpy as np
     from tensorflow.keras import backend as K
     from tensorflow.keras.models import Sequential
     from sklearn.metrics import classification_report, confusion_matrix


     print('Classification Report')
     target_names =['no', 'yes'] #change


     print(classification_report(test_generator.classes, y_pred, target_names=target_names))
```

```
Classification Report
              precision    recall  f1-score   support

          no       0.96      0.93      0.95       153
         yes       0.91      0.94      0.93       108

    accuracy                           0.94       261
   macro avg       0.94      0.94      0.94       261
weighted avg       0.94      0.94      0.94       261
```

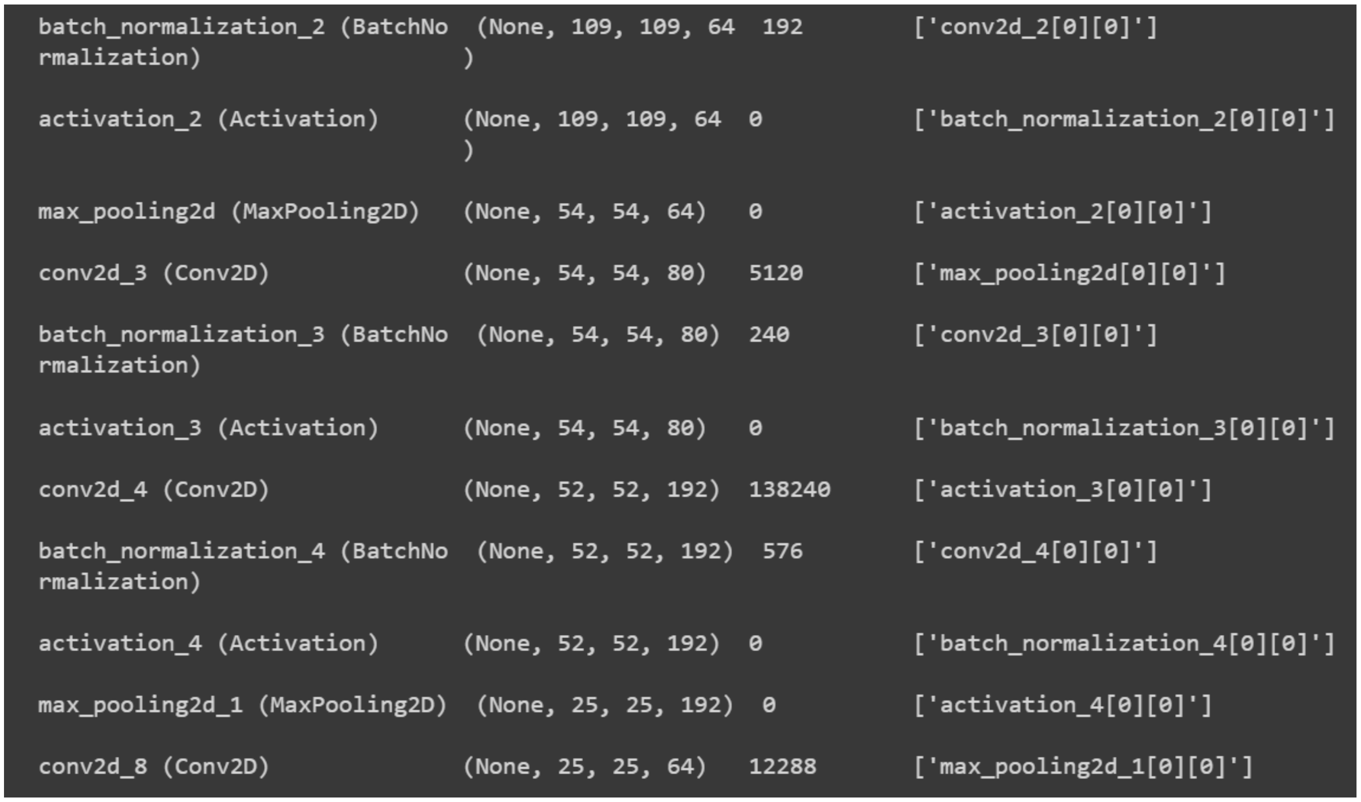
IMPLEMENTATION OF InceptionNetV3

The second model that we trained is InceptionNetV3 model. It is also a predefined deep learning model. At first all important libraries have been imported. After importing the libraries we import the data into google colab from google drive. After importing the dataset we augment the data so that the count of the data changes from 300 to 4000 images. After augmentation we create our model for for training and testing purposes. First we download the InceptionNetV3 deep learning model from the tensorflow library and then configure it to classify two classes. After creation of the model we compile the model. After compilation of the model we divide the dataset in the ration 80:10:10 ratio respectively. Then we fit the model fpr training. After training the model for 25-30 epochs. After training is completed we we create the confusion matrix and the classification report. The classification report tells us that the model has achieved an accuracy of 59%.

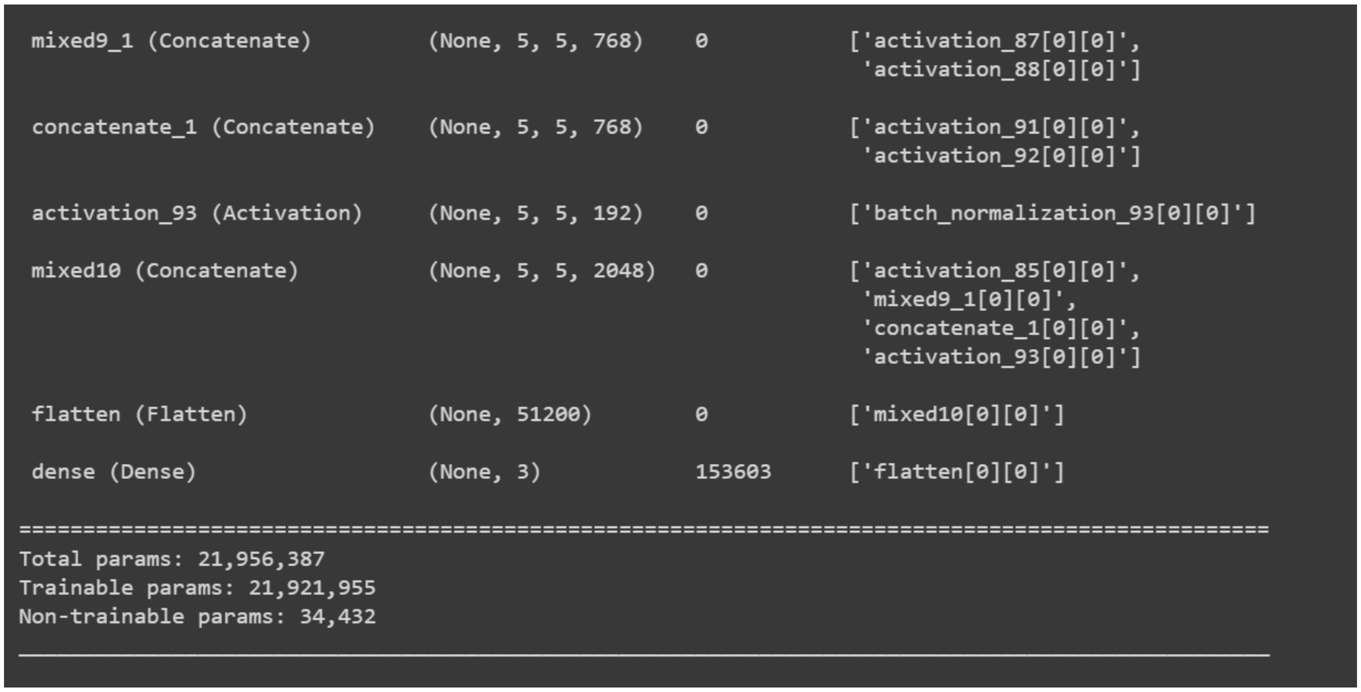
InceptionNetV3 model structure:



```
[ ] Model: "model"
    __________________________________________________________________________________________________
     Layer (type)                   Output Shape         Param #     Connected to
    ==================================================================================================
     input_1 (InputLayer)           [(None, 224, 224, 3  0           []
                                    )]

     conv2d (Conv2D)                (None, 111, 111, 32  864         ['input_1[0][0]']
                                    )

     batch_normalization (BatchNorm  (None, 111, 111, 32  96         ['conv2d[0][0]']
     alization)                     )

     activation (Activation)        (None, 111, 111, 32  0           ['batch_normalization[0][0]']
                                    )

     conv2d_1 (Conv2D)              (None, 109, 109, 32  9216        ['activation[0][0]']
                                    )

     batch_normalization_1 (BatchNo  (None, 109, 109, 32  96         ['conv2d_1[0][0]']
     rmalization)                   )

     activation_1 (Activation)      (None, 109, 109, 32  0           ['batch_normalization_1[0][0]']
                                    )

     conv2d_2 (Conv2D)              (None, 109, 109, 64  18432       ['activation_1[0][0]']
                                    )
```

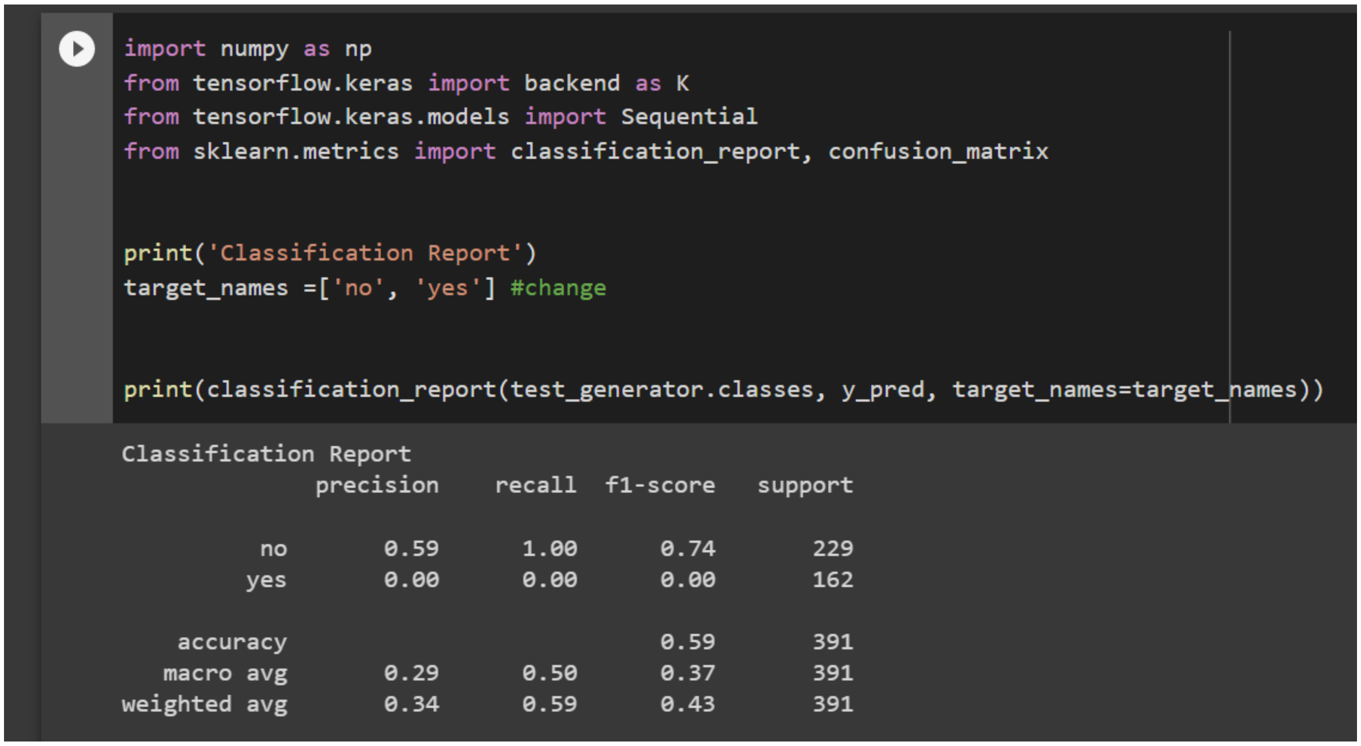


```
 batch_normalization_2 (BatchNo  (None, 109, 109, 64  192        ['conv2d_2[0][0]']
 rmalization)                   )

 activation_2 (Activation)      (None, 109, 109, 64  0           ['batch_normalization_2[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 54, 54, 64)   0           ['activation_2[0][0]']

 conv2d_3 (Conv2D)              (None, 54, 54, 80)   5120        ['max_pooling2d[0][0]']

 batch_normalization_3 (BatchNo  (None, 54, 54, 80)  240         ['conv2d_3[0][0]']
 rmalization)

 activation_3 (Activation)      (None, 54, 54, 80)   0           ['batch_normalization_3[0][0]']

 conv2d_4 (Conv2D)              (None, 52, 52, 192)  138240      ['activation_3[0][0]']

 batch_normalization_4 (BatchNo  (None, 52, 52, 192)  576        ['conv2d_4[0][0]']
 rmalization)

 activation_4 (Activation)      (None, 52, 52, 192)  0           ['batch_normalization_4[0][0]']

 max_pooling2d_1 (MaxPooling2D)  (None, 25, 25, 192)  0          ['activation_4[0][0]']

 conv2d_8 (Conv2D)              (None, 25, 25, 64)   12288       ['max_pooling2d_1[0][0]']
```



```
 mixed9_1 (Concatenate)         (None, 5, 5, 768)    0           ['activation_87[0][0]',
                                                                  'activation_88[0][0]']

 concatenate_1 (Concatenate)    (None, 5, 5, 768)    0           ['activation_91[0][0]',
                                                                  'activation_92[0][0]']

 activation_93 (Activation)     (None, 5, 5, 192)    0           ['batch_normalization_93[0][0]']

 mixed10 (Concatenate)          (None, 5, 5, 2048)   0           ['activation_85[0][0]',
                                                                  'mixed9_1[0][0]',
                                                                  'concatenate_1[0][0]',
                                                                  'activation_93[0][0]']

 flatten (Flatten)              (None, 51200)        0           ['mixed10[0][0]']

 dense (Dense)                  (None, 3)            153603      ['flatten[0][0]']

==================================================================================================
Total params: 21,956,387
Trainable params: 21,921,955
Non-trainable params: 34,432
__________________________________________________________________________________________________
```

Model accuracy:

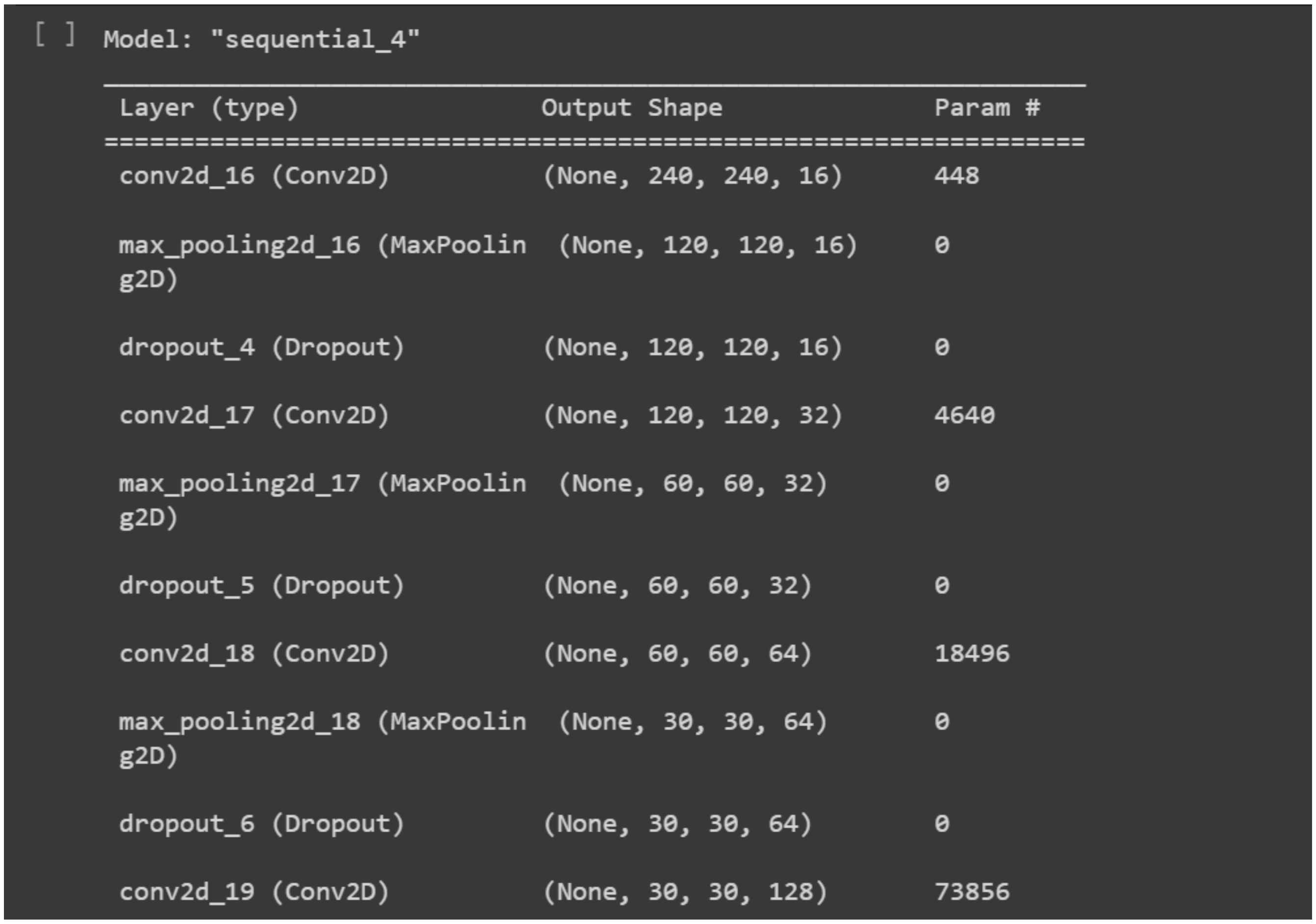

```
import numpy as np
from tensorflow.keras import backend as K
from tensorflow.keras.models import Sequential
from sklearn.metrics import classification_report, confusion_matrix


print('Classification Report')
target_names =['no', 'yes'] #change


print(classification_report(test_generator.classes, y_pred, target_names=target_names))
```

```
Classification Report
              precision    recall  f1-score   support

          no       0.59      1.00      0.74       229
         yes       0.00      0.00      0.00       162

    accuracy                           0.59       391
   macro avg       0.29      0.50      0.37       391
weighted avg       0.34      0.59      0.43       391
```

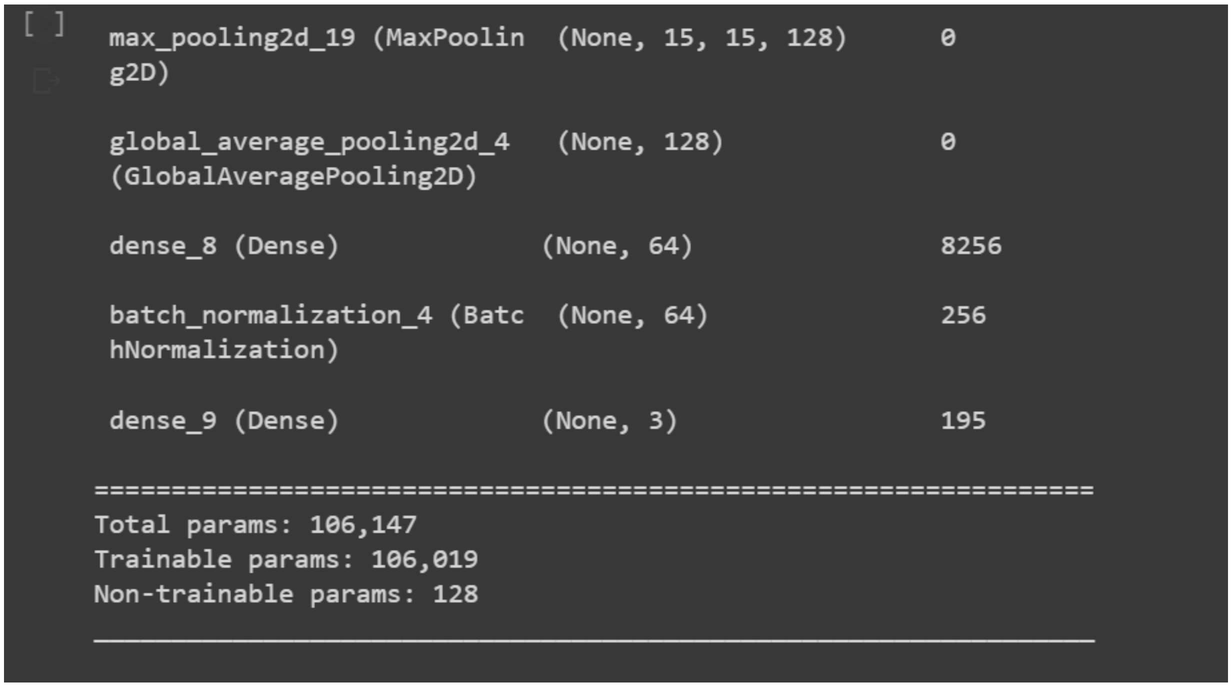
IMPLEMENTATION OF CUSTOM CNN

The third model that we trained is our custom CNN model. It is also a custom made deep learning model. At first all-important libraries have been imported. After importing the libraries, we import the data into google colab from google drive. After importing the dataset, we augment the data so that the count of the data changes from 300 t0 4000 images. After augmentation we create our model for for training and testing purposes. First, we download all the required layers from the tensorflow library required for model creation. We use normal convolution layers, dense layers, MaxPooling Layer, DropOut Layer and BatchNormalization Layer. We then create the model by adding the layers sequentially and then configure it to classify two classes. After creation of the model, we compile the model. After compilation of the model we divide the dataset in the ration 80:10:10 ratio respectively. Then we fit the model fpr training. After training the model for 25-30 epochs. After training is completed, we we create the confusion matrix and the classification report. The classification report tells us that the model has achieved an accuracy of 59%.

Model structure:

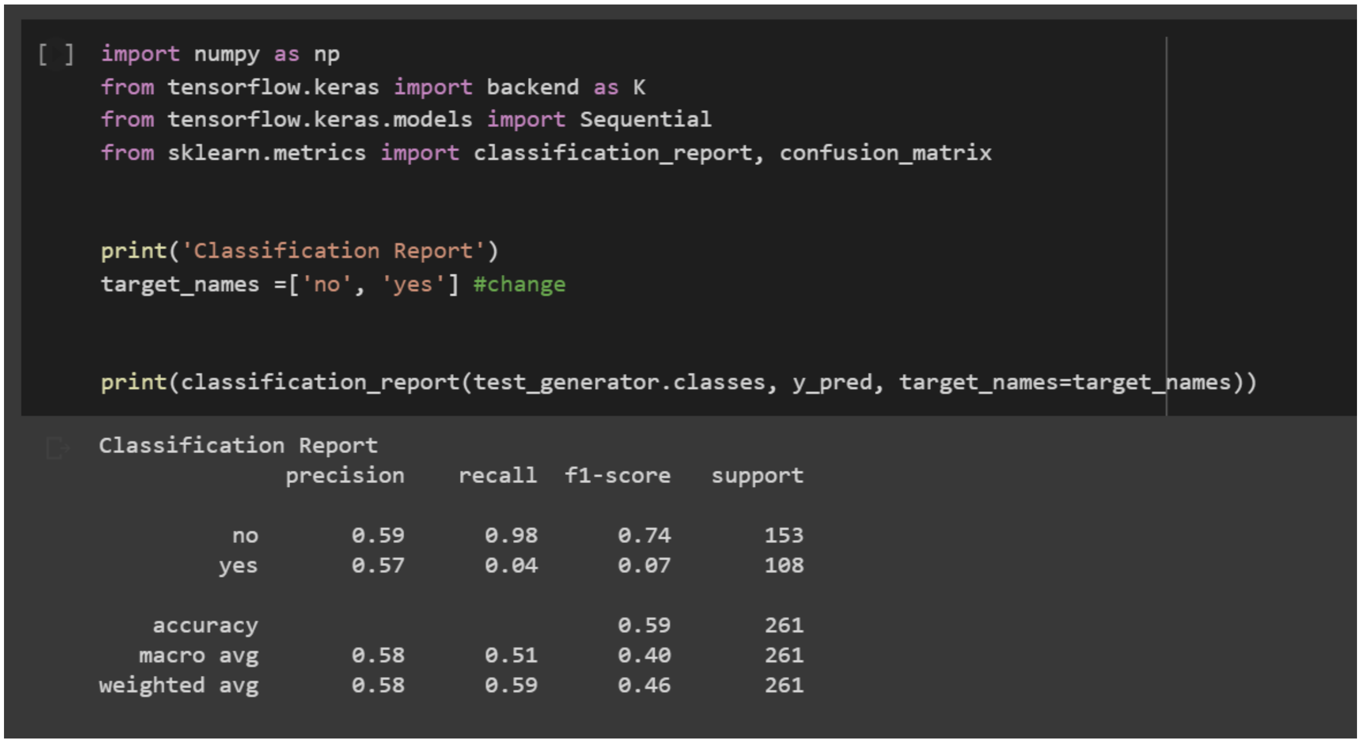


```
[ ] Model: "sequential_4"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     conv2d_16 (Conv2D)          (None, 240, 240, 16)      448

     max_pooling2d_16 (MaxPoolin  (None, 120, 120, 16)     0
     g2D)

     dropout_4 (Dropout)         (None, 120, 120, 16)      0

     conv2d_17 (Conv2D)          (None, 120, 120, 32)      4640

     max_pooling2d_17 (MaxPoolin  (None, 60, 60, 32)       0
     g2D)

     dropout_5 (Dropout)         (None, 60, 60, 32)        0

     conv2d_18 (Conv2D)          (None, 60, 60, 64)        18496

     max_pooling2d_18 (MaxPoolin  (None, 30, 30, 64)       0
     g2D)

     dropout_6 (Dropout)         (None, 30, 30, 64)        0

     conv2d_19 (Conv2D)          (None, 30, 30, 128)       73856
```



```
 max_pooling2d_19 (MaxPoolin  (None, 15, 15, 128)      0
 g2D)

 global_average_pooling2d_4   (None, 128)              0
 (GlobalAveragePooling2D)

 dense_8 (Dense)             (None, 64)                8256

 batch_normalization_4 (Batc  (None, 64)               256
 hNormalization)

 dense_9 (Dense)             (None, 3)                 195

=================================================================
Total params: 106,147
Trainable params: 106,019
Non-trainable params: 128
_________________________________________________________________
```

Model accuracy:



```
import numpy as np
from tensorflow.keras import backend as K
from tensorflow.keras.models import Sequential
from sklearn.metrics import classification_report, confusion_matrix


print('Classification Report')
target_names =['no', 'yes'] #change


print(classification_report(test_generator.classes, y_pred, target_names=target_names))
```

```
Classification Report
              precision    recall  f1-score   support

          no       0.59      0.98      0.74       153
         yes       0.57      0.04      0.07       108

    accuracy                           0.59       261
   macro avg       0.58      0.51      0.40       261
weighted avg       0.58      0.59      0.46       261
```

CONCLUSION

After narrowing down the scope of our brain tumor detection to just three most valid candidate models that were- InceptionetV3, MobileNetV2 and Convolutional Neural Network. We got an accuracy of 59 percent on both InceptionNetV3 and CNN, but got an accuracy of 94% on MobileNet V2. This also affirms our commitment in reducing complexity, performance requirements and size requirements for running the model.

FUTURE WORK

Since we achieved an high accuracy with MobileNet V2 model which is a relatively small sized model, we in the future can develop a mobile app which can be used to classify brain tumors. Although this will require high amount of customisation.

REFERENCES

[1] Kasban, Hany & El-bendary, Mohsen & Salama, Dina. (2015). “A Comparative Study of Medical Imaging Techniques”. InternationalJournal of Information Science and Intelligent System. 4. 37-58.J.Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol.2. Oxford: Clarendon, 1892, pp.68–73.

[2] D. Surya Prabha and J. Satheesh Kumar, “Performance Evaluation of Image Segmentation using Objective Methods”, Indian Journal of Science and Technology, Vol 9(8), February 2016.

[3] Brain Tumor: Statistics, Cancer.Net Editorial Board, 11/2017 (Accessed on 17th January 2019)

[4] Kavitha Angamuthu Rajasekaran and Chellamuthu Chinna Gounder, Advanced Brain Tumour Segmentation from MRI Images, 2018.

[5] General Information About Adult Brain Tumors". NCI. 14 April 2014. Archived from the original on 5 July 2014. Retrieved 8 June 2014. (Accessed on 11th January 2019)

[6] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

[7] B. Devkota, Abeer Alsadoon, P.W.C. Prasad, A. K. Singh, A. Elchouemi, “Image Segmentation for Early Stage Brain Tumor Detection using Mathematical Morphological Reconstruction,” 6th
International Conference on Smart Computing and Communications, ICSCC 2017, 7-8 December 2017, Kurukshetra, India.

[8] Song, Yantao & Ji, Zexuan & Sun, Quansen & Yuhui, Zheng. (2016). “A Novel Brain Tumor Segmentation from Multi-Modality MRI via A Level-Set-Based Model”. Journal of Signal Processing Systems. 87. 10.1007/s11265-016-1188-4.
[9] Ehab F. Badran, Esraa Galal Mahmoud, Nadder Hamdy, “An Algorithm for Detecting Brain Tumors in MRI Images”, 7th International Conference on Cloud Computing, Data Science &
Engineering - Confluence, 2017.
[10] Pei L, Reza SMS, Li W, Davatzikos C, Iftekharuddin KM. “Improved brain tumor segmentation by utilizing tumor growth model in longitudinal brain MRI”. Proc SPIE Int Soc Opt Eng. 2017.
[11] Dina Aboul Dahab, Samy S. A. Ghoniemy, Gamal M. Selim, “Automated Brain Tumor Detection and Identification using Image Processing and Probabilistic Neural Network Techniques”, IJIPVC,
Vol. 1, No. 2, pp. 1-8, 2012.
[12] Mohd Fauzi Othman, Mohd Ariffanan and Mohd Basri, “Probabilistic Neural Network for Brain Tumor Classification”, 2nd International Conference on Intelligent Systems, Modelling and Simulation, 2011.
[13] A. Rajendran, R. Dhanasekaran, “Fuzzy Clustering and Deformable Model for Tumor Segmentation on MRI Brain Image: A Combined Approach,” International Conference on Communication Technology and System Design 2011.
[14] Sobhaninia, Zahra & Rezaei, Safiyeh & Noroozi, Alireza & Ahmadi, Mehdi & Zarrabi, Hamidreza & Karimi, Nader & Emami, Ali & Samavi, Shadrokh. (2018). “Brain Tumor Segmentation Using Deep
Learning by Type Specific Sorting of Images”.
[15] Gupta, Gaurav and Vinay Singh. “Brain Tumor segmentation and classification using Fcm and support vector machine.” (2017).

[16] Anam Mustaqeem, Ali Javed, Tehseen Fatima, “An Efficient Brain Tumor Detection Algorithm Using Watershed & Thresholding Based Segmentation”, I.J. Image, Graphics and Signal Processing, 2012, 10, 34-39.
[17] Seetha, J & Selvakumar Raja, S. (2018). “Brain Tumor Classification Using Convolutional Neural Networks. Biomedical and Pharmacology Journal”. 11. 1457-1461. 10.13005/bpj/1511.
[18] Mariam Saii, Zaid Kraitem, “Automatic Brain tumor detection in MRI using image processing techniques”, Biomedical Statistics and Informatics, Vol. 2, No. 2, pp. 73-76, 2017.