**SETI(Search for Extra Terrestrial Intelligence) SIGNAL CLASIFICATION USING MACHINE LEARNING**

*Submitted in partial fulfillment of the requirements for the degree of*

**Bachelor of Technology**

in

**Computer Science and Engineering**

*by*

## SANJAY M S

**15BCE0517**

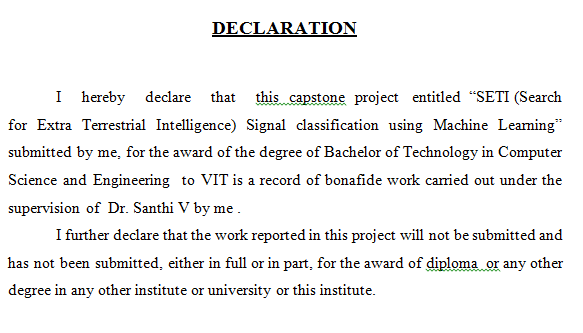
**Under the guidance of Dr. Santhi V.**

### School of Computing Science and Engineering

### 

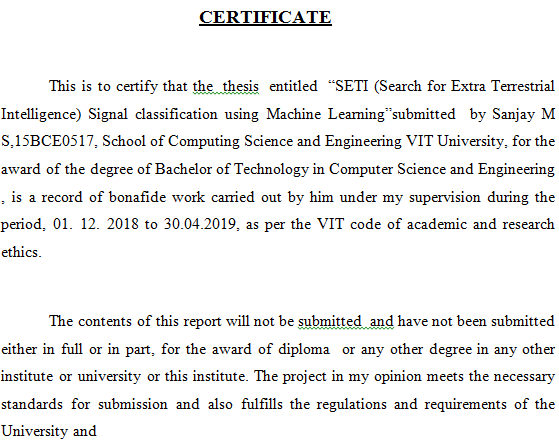
#### VIT, Vellore.

April, 2019



Place : Vellore Date :

#### Signature of the Candidate



Place : Vellore

Date : **Signature of the Guide**

#### Internal Examiner External Examiner

### Dr. Santhi V.

### Computer Science and Engineering

## ACKNOWLEDGEMENTS

I would like to thank my guide, my faculties and all other staffs of Vellore Institute of Technology, Vellore for their support and guidance without which this project would not have been possible.

#### Sanjay M S

# Summary

The project proposes to classify the spectrogram signal from space in order to detect signals from extra-terrestrial life forms. The spectrogram image is given as the input and image preprocessing techniques like image closing, edge detection using sobel operator are applied to remove noise and extract desired features. Then the input data is trained on standard CNN models like VGG16, VGG19, ALEXNET, GOOGLENET AND INCEPTION NET and their performances are compared to determine which model has a better accuracy. These are the standard CNN models and they are commonly used for image classification. Also a better model is proposed which classifies better than the considered standard models.

|  |  |  |
| --- | --- | --- |
| **Serial number** | **CONTENTS** | **Page number** |
|  | Acknowledgements i | i |
|  | Summary | ii |
|  | List of contents | iii |
|  | Figures list | iv |
|  | Tables list | v |
|  | Abbreviations list | vi |
| 1 | INTRODUCTION |  |
| 1.1 | Objective |  |
| 1.2 | Motivation |  |
| 1.3 | Background |  |
| 2 | PROJECT DESCRIPTION AND GOALS |  |
| 3 | TECHNICAL SPECIFICATION |  |
| 4 | DESIGN APPROACH AND DETAILS |  |
| 4.1 | Design Process & Methods |  |
| 4.2 | Codes and Standards |  |
| 4.3 | Constraints, Alternatives and Tradeoffs |  |
| 5 | SCHEDULE, TASKS AND MILESTONES |  |
| 6 | PROJECT DEMONSTRATION |  |
| 7 | RESULT & DISCUSSION |  |
| 8 | SUMMARY |  |
| 9 | REFERENCES |  |

**Figures list**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No.** |
| 1 | Model of VGG 16 |  |
| 2 | Model of VGG 16 |  |
| 3 | Model of VGG 16 |  |
| 4 | Model of Inception module |  |
| 5 | Model of Googlenet |  |
| 6 | Architecture of Googlenet |  |
| 7 | Model of Inception module A |  |
| 8 | Model of Inception net |  |
| 9 | Architectural diagram of proposed system |  |
| 10 | Class diagram |  |
| 11 | Use case diagram |  |
| 12 | Activity diagram |  |
| 13 | Component diagram |  |
| 14 | Gantt chart |  |
| 15 | Demonstration of VGG16 |  |
| 16 | Demonstration of VGG19 |  |
| 17 | Demonstration of Alexnet |  |
| 18 | Demonstration of Googlenet |  |
| 19 | Demonstration of Googlenet |  |
| 20 | Demonstration of Inception net |  |
| 21 | Demonstration of proposed model |  |
| 22 | Demonstration of proposed model |  |
| 23 | Accuracy of VGG16 |  |
| 24 | Loss of VGG16 |  |
| 25 | Accuracy of VGG19 |  |
| 26 | Loss of VGG19 |  |
| 27 | Accuracy of Alexnet |  |
| 28 | Loss of Alexnet |  |
| 29 | Accuracy of Googlenet |  |
| 30 | Loss of Googlenet |  |
| 31 | Accuracy of Inception net |  |
| 32 | Loss of Inception net |  |
| 33 | Accuracy of proposed model |  |
| 34 | Loss of Proposed model |  |

### Tables list

|  |  |  |
| --- | --- | --- |
| **Table No.** | **Title** | **Page No.** |
| 1. | Schedule |  |
| 2. | Requirement matrix |  |
| 3. | Classification report of VGG16 |  |
| 4. | Classification report of VGG19 |  |
| 5. | Classification report of Alexnet |  |
| 6, | Classification report of Googlenet |  |
| 7. | Classification report of Inception net |  |
| 8. | Classification report of proposed model |  |

## Abbreviations list

CNN Convolutional Neural Network

VGG Visual Geometry Group

ANN Artificial Neural Network

### 1. INTRODUCTION

1.1 OBJECTIVE

The aim of the proposed work is to classify spectrogram signals using machine learning techniques namely Convolutional Neural Network and Image processing techniques.

Other objectives of the proposed work are:

* Develop an algorithm to classify signals of the datasets.
* Stimulate the data to create new datasets with different types of signal.

1.2. MOTIVATION

During 2017, SETI conducted a machine learning competition where simulated datasets were given to the competitors and a blinded test set. The team that won the competition, obtained an accuracy of about 95 percent. They used a Convolutional Neural Network and the aim was to classify signals by going beyond the traditional methods for signal analysis. The goal was to convert the signal classification task into image classification by converting them into spectrograms. [1]

SETI has now started to release datasets for machine learning for the purpose of automated classification. Even though the dataset that is released is small, it is still sufficient for deep learning analysis. The motivation of this project is to apply machine learning techniques to classify images with decent accuracy.

1.3. BACKGROUND

The goal of SETI is to understand and explore the nature and origin of life in this universe and how intelligence evolved. The University of Berkeley, Berkley SETI Research Center created a project named “Breakthrough Listen” which is stated as one of the most elaborate search for alien life and communication. The radio signal data from space is gathered by Parkes Observatory in New South Wales and Green Bank Observatory in West Virginia and the optical data is collected by Automated Planet finder located in California

This project has the software and hardware for signal collection, money, time and the expert to run. The only sticking point is the data. Even after compromising on the raw data’s time or frequency resolution, Breakthrough Listen is archiving 500GB and data every hour[1]

**2. PROJECT DESCRIPTION AND GOALS**

The project proposes to classify the spectrogram signal from space in order to detect signals from extra-terrestrial life forms. The spectrogram image is given as the input and image preprocessing techniques like image closing, edge detection using sobel operator are applied to remove noise and extract desired features. Then the input data is trained on standard CNN models like VGG16, VGG19, ALEXNET, GOOGLENET AND INCEPTION NET and their performances are compared to determine which model has a better accuracy. These are the standard CNN models and they are commonly used for image classification. Also a better model is proposed which classifies better than the considered standard models.

The goal of the project is to develop an accurate and fat method for signal classification. These signals are persistent all the time and all days of the year. Also, these need to be processed instantly so that the movement of the signals can be detected. Thus automated classification is the best choice and human classification though possible, is clearly redundant. Convolutional Neural Network are the best suited for image classification. Especially since the dataset is large, Deep learning seemed to be a better option as it gives high accuracy. Initially, the standard models were tried out to determine if they were able to classify with a decent accuracy. But, the accuracy of these models were poor mainly because the ration of number of weights to the number of training data is very large i.e. the neural network is expected to train on a small dataset compared to the size of the network. So a simpler model is proposed for the classification to be better. It is also determined that a 3X3 mask in the Convolutional layer performs better than any other masks.

**3. TECHNICAL SPECIFICATION**

Convolutional Neural Networks, better Known as CNN (Conv Nets) are one of the premier, state of art, Artificial Neural Network design architecture, which helps in Image Based Classifications. The Basic Principle behind the working of CNN is the idea of Convolution,Which produces filtered Feature Maps stacked over each other.

Working:

Input Image in form of a Matrix, is convolved with m x m Filter (where m is generally Odd). Here a given Image might be Convolved with n different Filter of size m x m, where each of the filter would be focusing on a particular feature which needs to be extracted from the Input Image, thus forming n stacked Feature Maps. This Stacked Feature Maps are passed through a ReLu Layer, which removes all the Negative entries in the Matrices obtained, with value 0. Now the result of this is passed to Pooling Layer where we reduce the size of the obtained Matrices, by choosing a Nearest Neighbor Approach where an area of size p x p is replaced by the max value in the area p x p. The Purpose Pooling serves is: It reduces the size of the Matrix we Operate on, thus reducing the Computations, without loss of significant Information. It also helps in maintaining Translation and Rotational Invariance. The above Steps are repeated, and then at a certain stage the Matrices are vectorized and Sent to a Neural Network, which trains using Back-propagation, to classify the Image.

**VGG16:**

The image input to VGG16 is an image of dimension 224 X 224. The image is then processed to a set of convolutional layers whose dimensions are 3 X 3 i.e. the filters are very small and in fact the smallest one to detect the notions of up, down, center, left and right. Sometimes a linear 1X1 filter is also used followed by non-linear channels. The stride for the convolution is one pixel wide. The padding id designed in such a way that the dimensionality is preserved after the convolution i.e. dimensions don’t change. Pooling is done using max pooling and there are 5 max pool layer which follow convolutional layers. The dimension of the max pooling layer is 2X2 and the strie is 1 pixel.

The hidden layers have Rectified linear unit or ReLU for non-linear output. At the end of the model, 3 fully connected layers are present, out of which the first two have 4096 neurons and the last one has 1000 neurons for final classification.



**Fig. 1.** Model of VGG 16

**VGG19:**

The VGG19 is similar to VGG16 with 3 additionally stacked 3X3 Convolutional layers in the high layers.

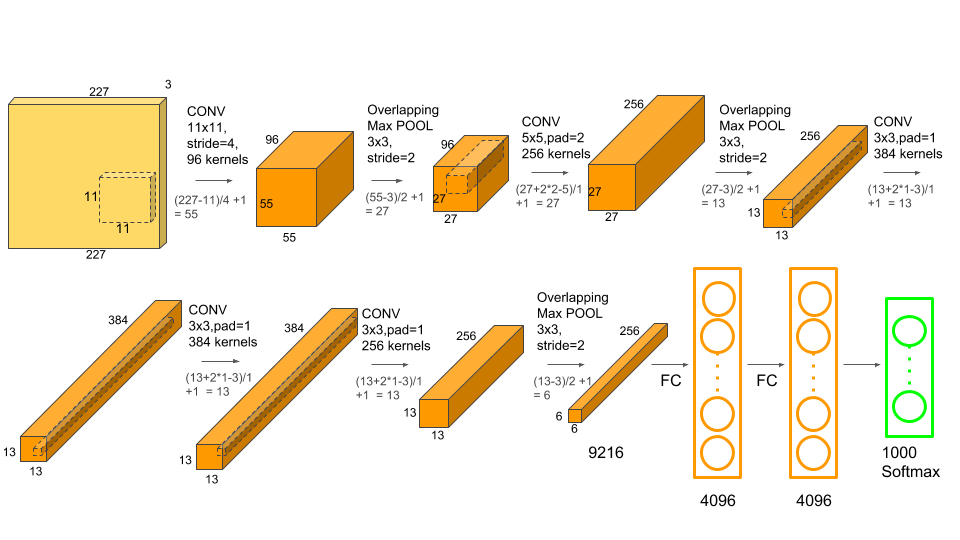


**Fig. 2.** Model of VGG 16

**Alexnet:**

The architecture of alexnet is made of 3 fully connected and 5 convolutional layers.

The first layer has 96 kernels, each of dimensions 11 X 11 X 3. After the first and second layer, there is a maxpooling layer which downsample the length and breadth of the convolutional layer. The window size is 3 X 3 and the stride is 2 pixels wide. The third, fourth and fifth convolutional layers are connected to each other and after the fifth layer, there is again a maxpooling layer. After that, there are 2 fully connected layers which finally leads to a softmax classifier for final output.

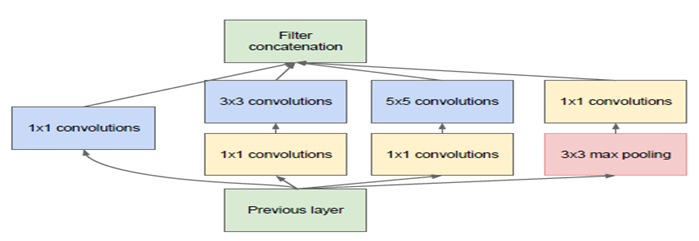


**Fig. 3.** Model of VGG 16

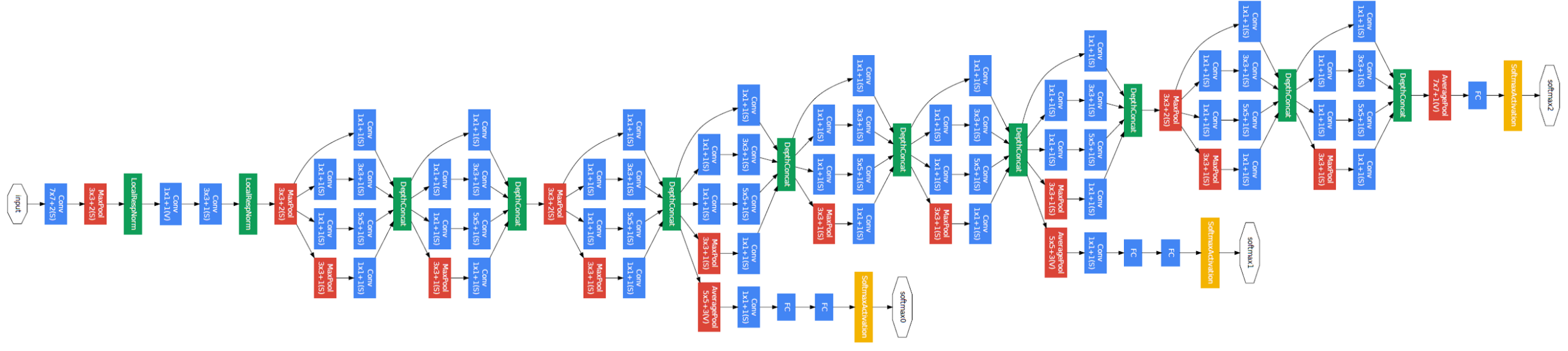
**Googlenet:**

The unique feature in Googlenet is that it has a 1 X 1convolution as well as inception modules and a global average pooling module. Other modules like maxpooling and convolutional layer as same as the other models.

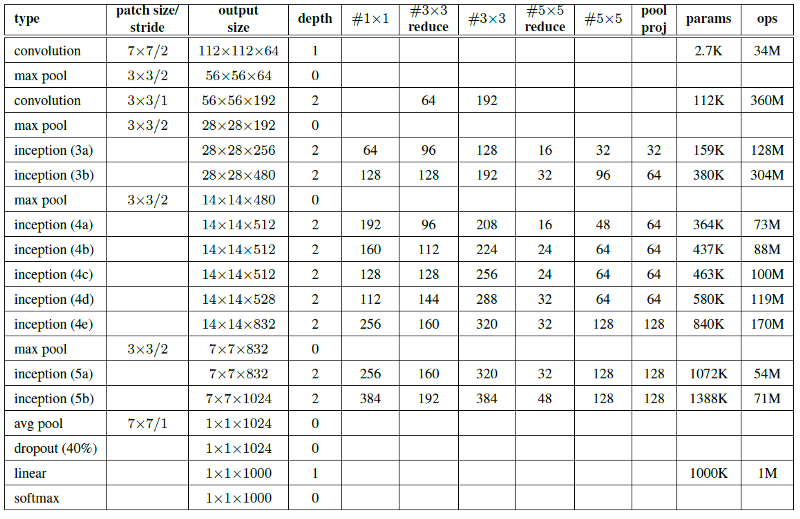
The 1X1 convolution layer is used along with Rectified Linear Unit(ReLU) so as to introduce non linearity. It also acts as a dimensionality reduction module. The Inception module is built using similar convolutional layers. One such example is:



**Fig. 4.** Model of Inception module



**Fig. 5.** Model of Googlenet

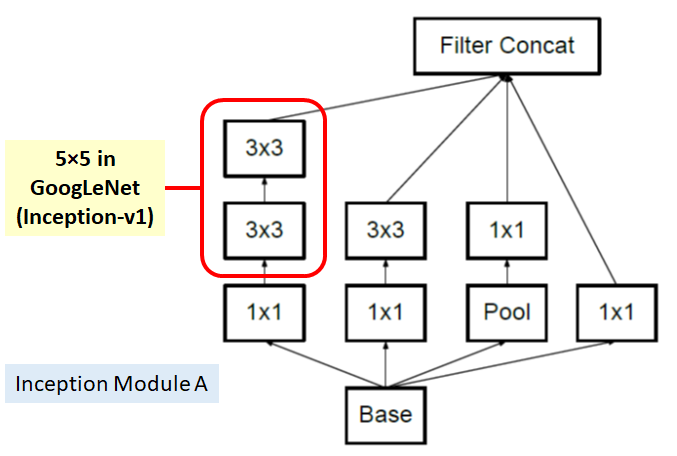


**Fig. 6.** Architecture of Googlenet

**Inception net(Version 3):**

Googlenet is often dubbed as Inception net Version 1. The third version is called Inception net. Several additions were made to Googlenet like batch normalization, factorisation ,auxiliary classifier and grid size reduction.

The factorization of convolutions involves replacing one high dimensional filter with two filters of smaller dimensions. The 5 X 5 large filter in Googlenet is replaced by two 3X3 filters. This reduces the number of connections without decreasing the accuracy.

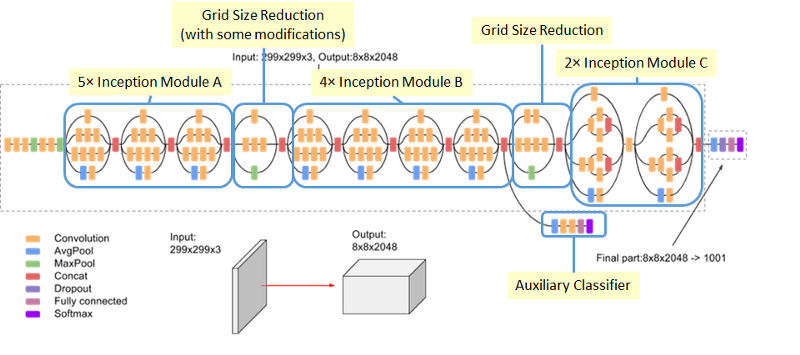


**Fig. 7.** Model of Inception module A

Also the 7 X 7 filter is replaced by two 1X7 and 7X1 filters.

There is only one auxiliary classifier used in Inception net instead of two layers.

Architecture:



**Fig. 8.** Model of Inception net

**4. DESIGN APPROACH AND DETAILS**

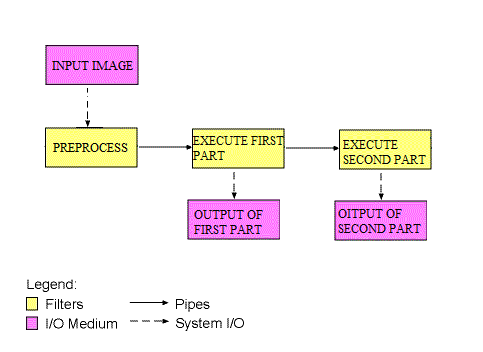
4.1 Design Process & Methods

This section gives a brief introduction and overview to the process of design. The system architecture is one way of approach to provide an all round view of the system and to provide a context for an external system. This helps the reader of the document and the user

to understand the design thoroughly as it gives a summary of the system before going to the depth of the design process.

.

4.2 High Level Design

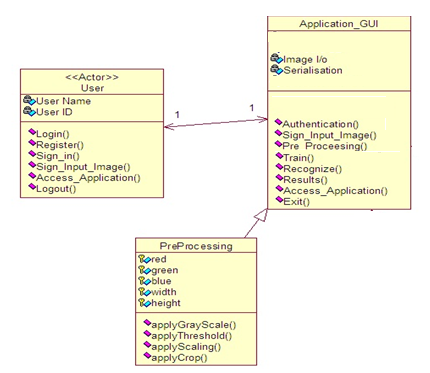


**Fig. 9.** Architectural diagram of proposed system

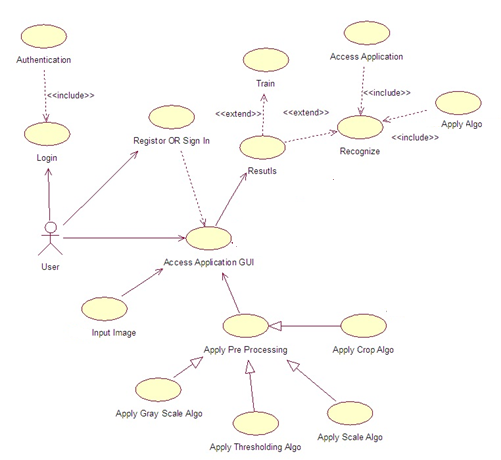
The architecture is pipe and filter model. In this type of model, there is a set of output and input stream, where one part of the system performs the reading of input data stream, one section processes the stream data and an other provides the output to the output stream component. Thus the name filters can be applied to the component. The connector component connects two streams, they transfer output from one to an another filter. Thus these connectors are called pipws, which transmits data from one place to another. The distinguished feature of this system is that filters are not dependent of any entity,i.e. they do not share any information to any other component. Also another important feature is that the filter doesn’t know to distinguish between upstream filter and downstream filter. They do not recognize the end components.

The software has basically 2 components, the first part does one vs all classification for one class. Based on the output of the first component, the second component is executed. If the output of first component is not a class, then the second component is executed. If not , the second component is not executed.

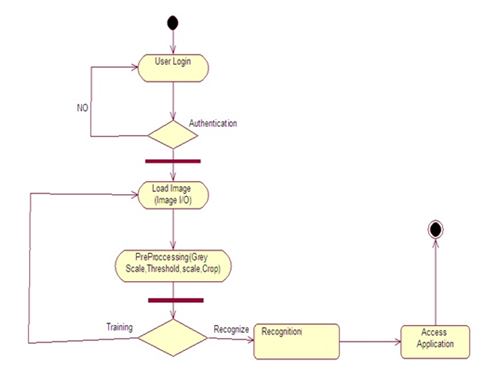
4.3 Detailed Design



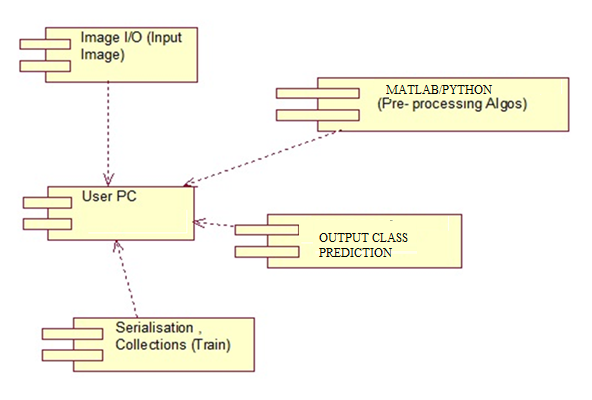
**Fig. 10.** Class diagram



**Fig. 11.** Use Case Diagram



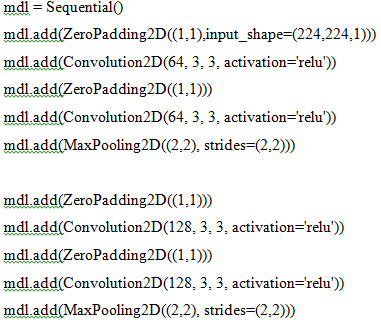
**Fig. 12.** Activity Diagram

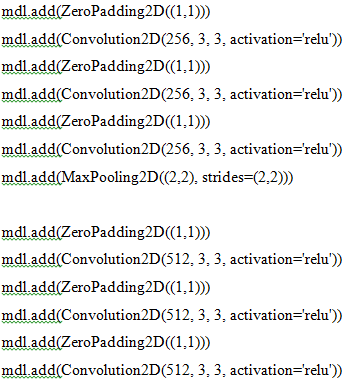


**Fig. 13.** Component Diagram

4.2 Codes and Standards .

**VGG 16:**



mdl.add(MaxPooling2D((2,2), strides=(2,2)))

mdl.add(ZeroPadding2D((1,1)))

mdl.add(Convolution2D(512, 3, 3, activation='relu'))

mdl.add(ZeroPadding2D((1,1)))

mdl.add(Convolution2D(512, 3, 3, activation='relu'))

mdl.add(ZeroPadding2D((1,1)))

mdl.add(Convolution2D(512, 3, 3, activation='relu'))

mdl.add(MaxPooling2D((2,2), strides=(2,2)))

mdl.add(Flatten())

mdl.add(Dense(4096, activation='relu'))

mdl.add(Dropout(0.5))

mdl.add(Dense(4096, activation='relu'))

mdl.add(Dropout(0.5))

mdl.add(Dense(7, activation='softmax'))

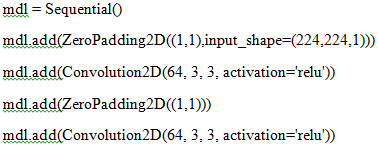
opt=Adam(lr=0.00146)

mdl.compile(optimizer=opt, loss='categorical\_crossentropy', metrics=["accuracy"])

print(mdl.summary())

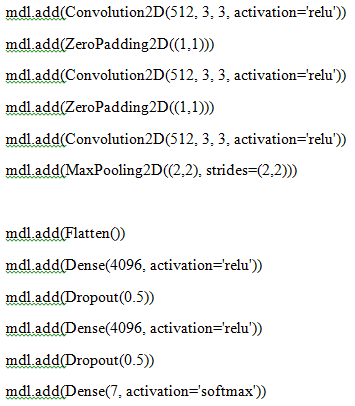
his = mdl.fit\_generator(train,epochs=10,steps\_per\_epoch=100,shuffle=True,verbose=1,validation\_data=(x\_valid, y\_valid))

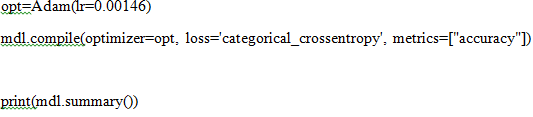
**VGG19:**







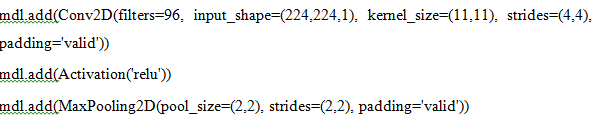




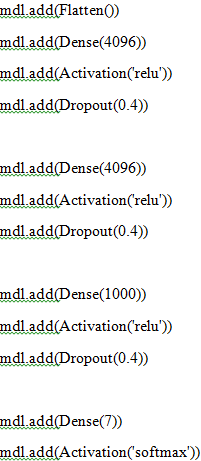
his = mdl.fit\_generator(train,epochs=10,steps\_per\_epoch=100,shuffle=True,verbose=1,validation\_data=(x\_valid, y\_valid))

**Alexnet**:

mdl = Sequential()







mdl.compile(Adam(lr=0.00146), loss="categorical\_crossentropy", metrics=["accuracy"])

print(mdl.summary())

his = mdl.fit\_generator(train,epochs=10,steps\_per\_epoch=100,validation\_data=(x\_valid, y\_valid),shuffle=True,verbose=1)

**Googlenet:**

from keras.layers import Conv2D, MaxPool2D,Dropout, Dense, Input, concatenate,GlobalAveragePooling2D, AveragePooling2D,Flatten

import keras

from keras.layers.core import Layer

import keras.backend as K

import tensorflow as tf

from keras.datasets import cifar10

import cv2

import numpy as np

from keras.datasets import cifar10

from keras import backend as K

from keras.utils import np\_utils

import math

from keras.optimizers import SGD

from keras.callbacks import LearningRateScheduler

def inception\_module(x,

filters\_1x1,

filters\_3x3\_reduce,

filters\_3x3,

filters\_5x5\_reduce,

filters\_5x5,

filters\_pool\_proj,

name=None):

conv\_1x1 = Conv2D(filters\_1x1, (1, 1), padding='same', activation='relu', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(x)

conv\_3x3 = Conv2D(filters\_3x3\_reduce, (1, 1), padding='same', activation='relu', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(x)

conv\_3x3 = Conv2D(filters\_3x3, (3, 3), padding='same', activation='relu', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(conv\_3x3)

conv\_5x5 = Conv2D(filters\_5x5\_reduce, (1, 1), padding='same', activation='relu', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(x)

conv\_5x5 = Conv2D(filters\_5x5, (5, 5), padding='same', activation='relu', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(conv\_5x5)

pool\_proj = MaxPool2D((3, 3), strides=(1, 1), padding='same')(x)

pool\_proj = Conv2D(filters\_pool\_proj, (1, 1), padding='same', activation='relu', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(pool\_proj)

output = concatenate([conv\_1x1, conv\_3x3, conv\_5x5, pool\_proj], axis=3, name=name)

return output

kernel\_init = keras.initializers.glorot\_uniform()

bias\_init = keras.initializers.Constant(value=0.2)

input\_layer = Input(shape=(224, 224, 1))

x = Conv2D(64, (7, 7), padding='same', strides=(2, 2), activation='relu', name='conv\_1\_7x7/2', kernel\_initializer=kernel\_init, bias\_initializer=bias\_init)(input\_layer)

x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max\_pool\_1\_3x3/2')(x)

x = Conv2D(64, (1, 1), padding='same', strides=(1, 1), activation='relu', name='conv\_2a\_3x3/1')(x)

x = Conv2D(192, (3, 3), padding='same', strides=(1, 1), activation='relu', name='conv\_2b\_3x3/1')(x)

x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max\_pool\_2\_3x3/2')(x)

x = inception\_module(x,

filters\_1x1=64,

filters\_3x3\_reduce=96,

filters\_3x3=128,

filters\_5x5\_reduce=16,

filters\_5x5=32,

filters\_pool\_proj=32,

name='inception\_3a')

x = inception\_module(x,

filters\_1x1=128,

filters\_3x3\_reduce=128,

filters\_3x3=192,

filters\_5x5\_reduce=32,

filters\_5x5=96,

filters\_pool\_proj=64,

name='inception\_3b')

x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max\_pool\_3\_3x3/2')(x)

x = inception\_module(x,

filters\_1x1=192,

filters\_3x3\_reduce=96,

filters\_3x3=208,

filters\_5x5\_reduce=16,

filters\_5x5=48,

filters\_pool\_proj=64,

name='inception\_4a')

x1 = AveragePooling2D((5, 5), strides=3)(x)

x1 = Conv2D(128, (1, 1), padding='same', activation='relu')(x1)

x1 = Flatten()(x1)

x1 = Dense(1024, activation='relu')(x1)

x1 = Dropout(0.7)(x1)

x1 = Dense(7, activation='softmax', name='auxilliary\_output\_1')(x1)

x = inception\_module(x,

filters\_1x1=160,

filters\_3x3\_reduce=112,

filters\_3x3=224,

filters\_5x5\_reduce=24,

filters\_5x5=64,

filters\_pool\_proj=64,

name='inception\_4b')

x = inception\_module(x,

filters\_1x1=128,

filters\_3x3\_reduce=128,

filters\_3x3=256,

filters\_5x5\_reduce=24,

filters\_5x5=64,

filters\_pool\_proj=64,

name='inception\_4c')

x = inception\_module(x,

filters\_1x1=112,

filters\_3x3\_reduce=144,

filters\_3x3=288,

filters\_5x5\_reduce=32,

filters\_5x5=64,

filters\_pool\_proj=64,

name='inception\_4d')

x2 = AveragePooling2D((5, 5), strides=3)(x)

x2 = Conv2D(128, (1, 1), padding='same', activation='relu')(x2)

x2 = Flatten()(x2)

x2 = Dense(1024, activation='relu')(x2)

x2 = Dropout(0.7)(x2)

x2 = Dense(7, activation='softmax', name='auxilliary\_output\_2')(x2)

x = inception\_module(x,

filters\_1x1=256,

filters\_3x3\_reduce=160,

filters\_3x3=320,

filters\_5x5\_reduce=32,

filters\_5x5=128,

filters\_pool\_proj=128,

name='inception\_4e')

x = MaxPool2D((3, 3), padding='same', strides=(2, 2), name='max\_pool\_4\_3x3/2')(x)

x = inception\_module(x,

filters\_1x1=256,

filters\_3x3\_reduce=160,

filters\_3x3=320,

filters\_5x5\_reduce=32,

filters\_5x5=128,

filters\_pool\_proj=128,

name='inception\_5a')

x = inception\_module(x,

filters\_1x1=384,

filters\_3x3\_reduce=192,

filters\_3x3=384,

filters\_5x5\_reduce=48,

filters\_5x5=128,

filters\_pool\_proj=128,

name='inception\_5b')

x = GlobalAveragePooling2D(name='avg\_pool\_5\_3x3/1')(x)

x = Dropout(0.4)(x)

x = Dense(7, activation='softmax', name='output')(x)

from keras.model import Mdl

mdl = Mdl(input\_layer, [x, x1, x2], name='inception\_v1')

mdl.summary()

opt=Adam(lr=0.00146)

mdl.compile(optimizer=opt, loss='categorical\_crossentropy', metrics=["accuracy"])

his=mdl.fit(b,[c,c,c],epochs=10,shuffle=True,verbose=1,validation\_data=(x\_valid,[y\_valid,y\_valid,y\_valid]))

**Inception net:**

def conv2d\_bn(x,

filters,

num\_row,

num\_col,

padding='same',

strides=(1, 1),

name=None):

if name is not None:

bn\_name = name + '\_bn'

conv\_name = name + '\_conv'

else:

bn\_name = None

conv\_name = None

bn\_axis = 3

x = Conv2D(

filters, (num\_row, num\_col),

strides=strides,

padding=padding,

use\_bias=False,

name=conv\_name)(x)

x = BatchNormalization(axis=bn\_axis, scale=False, name=bn\_name)(x)

x = Activation('relu', name=name)(x)

return x

img\_input = Input(shape=(224,224,1))

channel\_axis = 3

x = conv2d\_bn(img\_input, 32, 3, 3, strides=(2, 2), padding='valid')

x = conv2d\_bn(x, 32, 3, 3, padding='valid')

x = conv2d\_bn(x, 64, 3, 3)

x = MaxPooling2D((3, 3), strides=(2, 2))(x)

x = conv2d\_bn(x, 80, 1, 1, padding='valid')

x = conv2d\_bn(x, 192, 3, 3, padding='valid')

x = MaxPooling2D((3, 3), strides=(2, 2))(x)

# mixed 0: 35 x 35 x 256

branch1x1 = conv2d\_bn(x, 64, 1, 1)

branch5x5 = conv2d\_bn(x, 48, 1, 1)

branch5x5 = conv2d\_bn(branch5x5, 64, 5, 5)

branch3x3dbl = conv2d\_bn(x, 64, 1, 1)

branch3x3dbl = conv2d\_bn(branch3x3dbl, 128, 3, 3)

branch3x3dbl = conv2d\_bn(branch3x3dbl, 256, 3, 3)

branch\_pool = AveragePooling2D((3, 3),strides=(1, 1),padding='same')(x)

branch\_pool = conv2d\_bn(branch\_pool, 32, 1, 1)

x = concatenate([branch1x1, branch5x5, branch3x3dbl, branch\_pool],axis=channel\_axis,name='mixed0')

# mixed 1: 35 x 35 x 288

branch1x1 = conv2d\_bn(x, 64, 1, 1)

branch5x5 = conv2d\_bn(x, 48, 1, 1)

branch5x5 = conv2d\_bn(branch5x5, 64, 5, 5)

branch3x3dbl = conv2d\_bn(x, 64, 1, 1)

branch3x3dbl = conv2d\_bn(branch3x3dbl, 512, 3, 3)

branch\_pool = AveragePooling2D((3, 3),strides=(1, 1),padding='same')(x)

branch\_pool = conv2d\_bn(branch\_pool, 64, 1, 1)

x =concatenate([branch1x1, branch5x5, branch3x3dbl, branch\_pool],axis=channel\_axis,name='mixed1')

# mixed 2: 35 x 35 x 288

branch1x1 = conv2d\_bn(x, 64, 1, 1)

branch5x5 = conv2d\_bn(x, 48, 1, 1)

branch5x5 = conv2d\_bn(branch5x5, 64, 5, 5)

branch3x3dbl = conv2d\_bn(x, 64, 1, 1)

branch3x3dbl = conv2d\_bn(branch3x3dbl, 128, 3, 3)

branch\_pool = AveragePooling2D((3, 3),strides=(1, 1),padding='same')(x)

branch\_pool = conv2d\_bn(branch\_pool, 64, 1, 1)

x = concatenate([branch1x1, branch5x5, branch3x3dbl, branch\_pool],axis=channel\_axis,name='mixed2')

# mixed 3: 17 x 17 x 768

branch3x3 = conv2d\_bn(x, 384, 3, 3, strides=(2, 2), padding='valid')

branch3x3dbl = conv2d\_bn(x, 64, 1, 1)

branch3x3dbl = conv2d\_bn(branch3x3dbl, 512, 3, 3, strides=(2, 2), padding='valid')

branch\_pool = MaxPooling2D((3, 3), strides=(2, 2))(x)

x = concatenate([branch3x3, branch3x3dbl, branch\_pool],axis=channel\_axis,name='mixed3')

# mixed 4: 17 x 17 x 768

branch1x1 = conv2d\_bn(x, 192, 1, 1)

branch7x7 = conv2d\_bn(x, 128, 1, 1)

branch7x7 = conv2d\_bn(branch7x7, 128, 1, 7)

branch7x7 = conv2d\_bn(branch7x7, 192, 7, 1)

branch7x7dbl = conv2d\_bn(x, 128, 1, 1)

branch7x7dbl = conv2d\_bn(branch7x7dbl, 128, 7, 1)

branch7x7dbl = conv2d\_bn(branch7x7dbl, 128, 1, 7)

branch7x7dbl = conv2d\_bn(branch7x7dbl, 128, 7, 1)

branch7x7dbl = conv2d\_bn(branch7x7dbl, 192, 1, 7)

branch\_pool = AveragePooling2D((3, 3),strides=(1, 1),padding='same')(x)

branch\_pool = conv2d\_bn(branch\_pool, 192, 1, 1)

x = concatenate([branch1x1, branch7x7, branch7x7dbl, branch\_pool],axis=channel\_axis,name='mixed4')

# mixed 5, 6: 17 x 17 x 768

for i in range(2):

branch1x1 = conv2d\_bn(x, 192, 1, 1)

branch7x7 = conv2d\_bn(x, 160, 1, 1)

branch7x7 = conv2d\_bn(branch7x7, 160, 1, 7)

branch7x7 = conv2d\_bn(branch7x7, 192, 7, 1)

branch7x7dbl = conv2d\_bn(x, 160, 1, 1)

branch7x7dbl = conv2d\_bn(branch7x7dbl, 160, 7, 1)

branch7x7dbl = conv2d\_bn(branch7x7dbl, 160, 1, 7)

branch7x7dbl = conv2d\_bn(branch7x7dbl, 160, 7, 1)

branch7x7dbl = conv2d\_bn(branch7x7dbl, 192, 1, 7)

branch\_pool = AveragePooling2D((3, 3), strides=(1, 1), padding='same')(x)

branch\_pool = conv2d\_bn(branch\_pool, 192, 1, 1)

x = concatenate([branch1x1, branch7x7, branch7x7dbl, branch\_pool],axis=channel\_axis,name='mixed' + str(5 + i))

# mixed 7: 17 x 17 x 768

branch1x1 = conv2d\_bn(x, 192, 1, 1)

branch7x7 = conv2d\_bn(x, 192, 1, 1)

branch7x7 = conv2d\_bn(branch7x7, 192, 1, 7)

branch7x7 = conv2d\_bn(branch7x7, 192, 7, 1)

branch7x7dbl = conv2d\_bn(x, 192, 1, 1)

branch7x7dbl = conv2d\_bn(branch7x7dbl, 192, 7, 1)

branch7x7dbl = conv2d\_bn(branch7x7dbl, 192, 1, 7)

branch7x7dbl = conv2d\_bn(branch7x7dbl, 192, 7, 1)

branch7x7dbl = conv2d\_bn(branch7x7dbl, 192, 1, 7)

branch\_pool = AveragePooling2D((3, 3),

strides=(1, 1),

padding='same')(x)

branch\_pool = conv2d\_bn(branch\_pool, 192, 1, 1)

x = concatenate(

[branch1x1, branch7x7, branch7x7dbl, branch\_pool],

axis=channel\_axis,

name='mixed7')

# mixed 8: 8 x 8 x 1280

branch3x3 = conv2d\_bn(x, 192, 1, 1)

branch3x3 = conv2d\_bn(branch3x3, 320, 3, 3,

strides=(2, 2), padding='valid')

branch7x7x3 = conv2d\_bn(x, 192, 1, 1)

branch7x7x3 = conv2d\_bn(branch7x7x3, 192, 1, 7)

branch7x7x3 = conv2d\_bn(branch7x7x3, 192, 7, 1)

branch7x7x3 = conv2d\_bn(

branch7x7x3, 192, 3, 3, strides=(2, 2), padding='valid')

branch\_pool = MaxPooling2D((3, 3), strides=(2, 2))(x)

x = concatenate(

[branch3x3, branch7x7x3, branch\_pool],

axis=channel\_axis,

name='mixed8')

# mixed 9: 8 x 8 x 2048

for i in range(1):

branch1x1 = conv2d\_bn(x, 320, 1, 1)

branch3x3 = conv2d\_bn(x, 384, 1, 1)

branch3x3\_1 = conv2d\_bn(branch3x3, 384, 1, 3)

branch3x3\_2 = conv2d\_bn(branch3x3, 384, 3, 1)

branch3x3 = concatenate(

[branch3x3\_1, branch3x3\_2],

axis=channel\_axis,

name='mixed9\_' + str(i))

branch3x3dbl = conv2d\_bn(x, 448, 1, 1)

branch3x3dbl = conv2d\_bn(branch3x3dbl, 384, 3, 3)

branch3x3dbl\_1 = conv2d\_bn(branch3x3dbl, 384, 1, 3)

branch3x3dbl\_2 = conv2d\_bn(branch3x3dbl, 384, 3, 1)

branch3x3dbl = concatenate(

[branch3x3dbl\_1, branch3x3dbl\_2], axis=channel\_axis)

branch\_pool = AveragePooling2D(

(3, 3), strides=(1, 1), padding='same')(x)

branch\_pool = conv2d\_bn(branch\_pool, 192, 1, 1)

x = concatenate(

[branch1x1, branch3x3, branch3x3dbl, branch\_pool],

axis=channel\_axis,

name='mixed' + str(9 + i))

# Classification block

x = GlobalAveragePooling2D(name='avg\_pool')(x)

x = Dense(7, activation='softmax', name='predictions')(x)

mdl=Mdl(img\_input, x, name='inception\_v3')

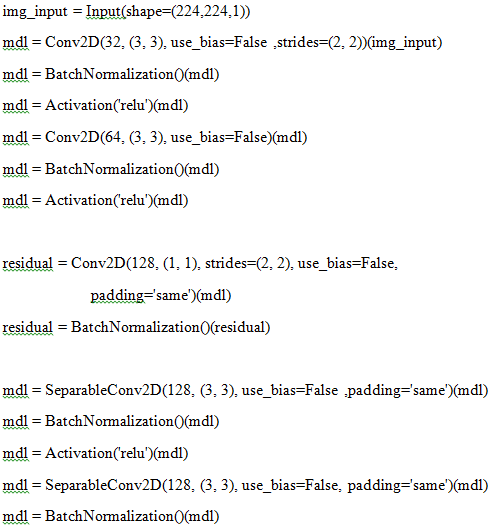
mdl.summary()

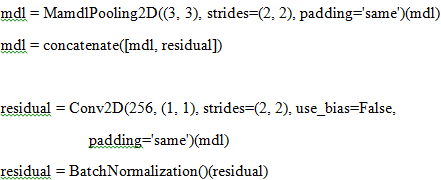
opt=Adam(lr=0.0015)

mdl.compile(optimizer=opt, loss='categorical\_crossentropy', metrics=["accuracy"])

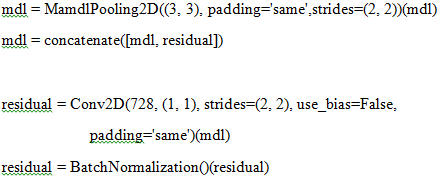
his = mdl.fit\_generator(train,epochs=50,steps\_per\_epoch=100,validation\_data=(x\_valid, y\_valid),shuffle=True,verbose=1)

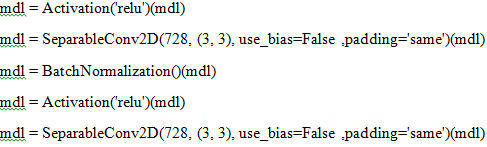
**Proposed model:**

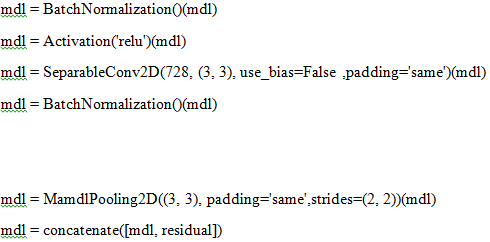


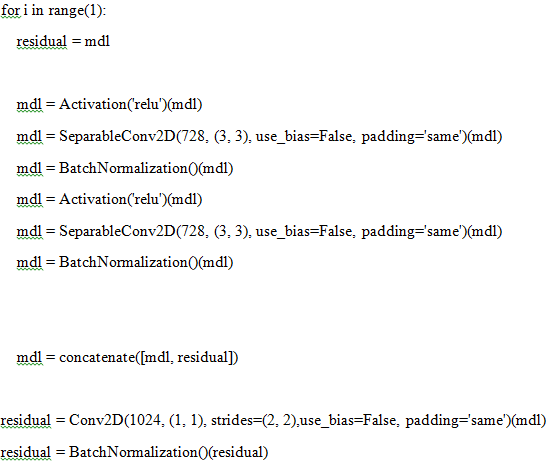


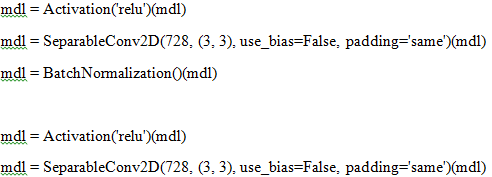




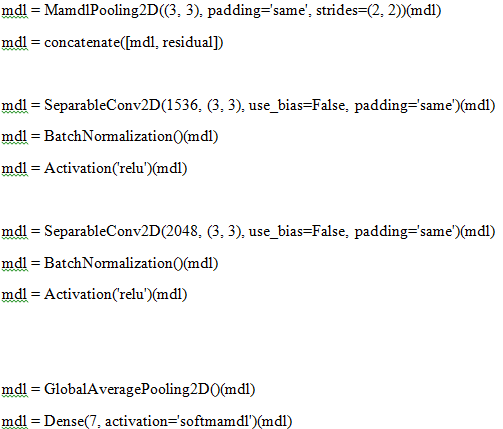








mdl = BatchNormalization()(mdl)



mdl = Mdl(img\_input, mdl, name='mdlception')

mdl.summary()

opt=Adam(lr=0.0015)

mdl.compile(optimizer=opt, loss='categorical\_crossentropy', metrics=["accuracy"])

his = mdl.fit\_generator(train,epochs=30,steps\_per\_epoch=100,validation\_data=(mdl\_valid, y\_valid),shuffle=True,verbose=1)

4.3 Constraints, Alternatives and Tradeoffs

The main constraint was that the predefined models all had around 1000 output classes whereas the project required only 7 classes. So the final layer alone is modified to suit the needs. The system can run only on 64 bit system.

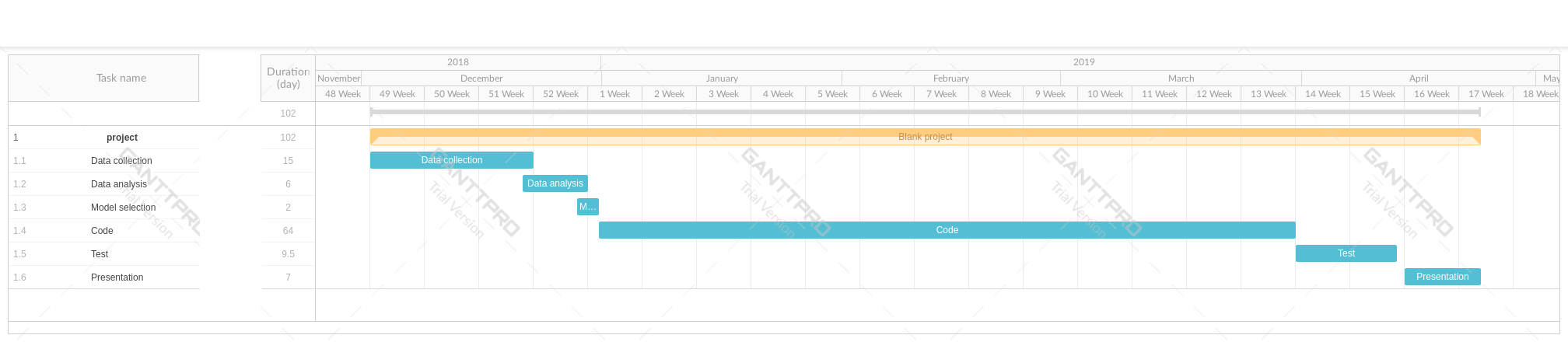
The system assumes that a GPU(Graphics Processing Unit) is installed in the system. Also,t he system requires keras python library and keras works only on 64 bit system. Matlab is also a pre-requisite.

Another assumption made is that the system works perfectly and does not malfunction.

Dependencies:

The system is divided into two parts. The output of the first part determines whether the second part is required to be executed. The first part classifies one among the seven output classes(one vs all classification). If it classifies accordingly, second part need not be executed as the output class is already known. If not, then the second part is to be executed to know the class.

**5. SCHEDULE,TASKS AND MILESTONES**



**Fig. 14.** Gantt chart

**Table 1.** Schedule

| ***Activity*** | ***Planned***  ***Completion***  ***Date*** | ***Actual Completion Date*** | ***Deliverable/ Checkpoint*** |
| --- | --- | --- | --- |
| *Identify data* | *3/12/18* | *21/12/18* | *Dataset* |
| *Analysis of data* | *21/12/18* | *28/12/18* | *Properties of image, type of classes, problem formulation* |
| *Model selection* | *28/12/18* | *31/12/18* | *Models for classification* |
| *Coding* | *1/1/19* | *29/3/19* | *Python code and result of each model* |
| *Testing phase* | *1/4/19* | *12/4/19* | *Verification of code and results* |
| *Final presentation and documentation* | *15/4/19* | *23/4/19* | *Document and poster preparation* |

**Table 2.** Requirement matrix

|  |  |
| --- | --- |
| **ID** | **Functional Requirements** |
| R01 | The system should be able to take any number of arguments from the user |
| R02 | The system should produce the output in a definite time |
| R03 | The system should have the appropriate dataset |
| R04 | The system should output only one class |
| R05 | The system should prevent overfitting of data |
| R06 | The system should not tolerate high error rates |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Cases** | **R01** | **R02** | **R03** | **R04** | **R05** | **R06** |
| **T01** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **T02** | ✓ | ✓ |  |  |  |  |
| **T03** |  |  |  | ✓ | ✓ |  |
| **T04** |  |  |  |  | ✓ | ✓ |

T01: verify the user is able to interact with the system.

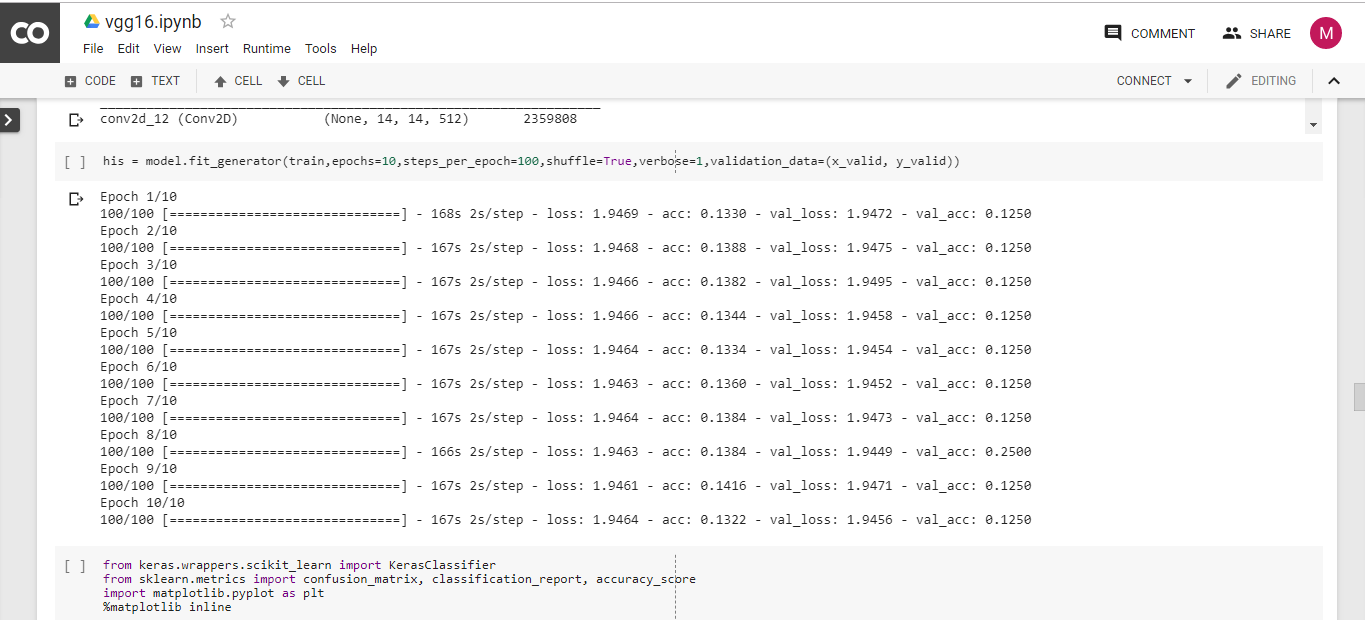
T02: Verify whether input and output are accessible.

T03:Verify whether data is available and usable.

T04:Verify whether system is accurate enough to provide a considerable output.

**6. PROJECT DEMONSTRATION**

VGG 16:



**Fig. 15.** Demonstration of VGG16

VGG 19:



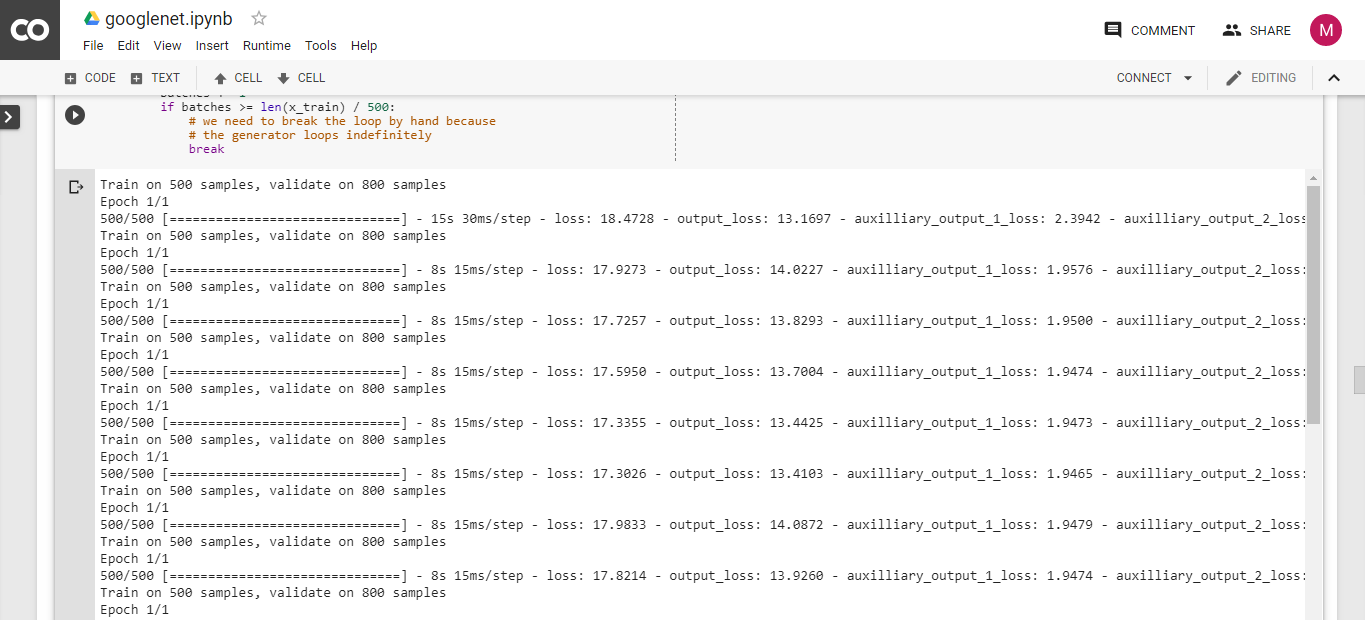
**Fig. 16.** Demonstration of VGG19

ALEXNET:



**Fig. 17.** Demonstration of Alexnet

GOOGLENET:

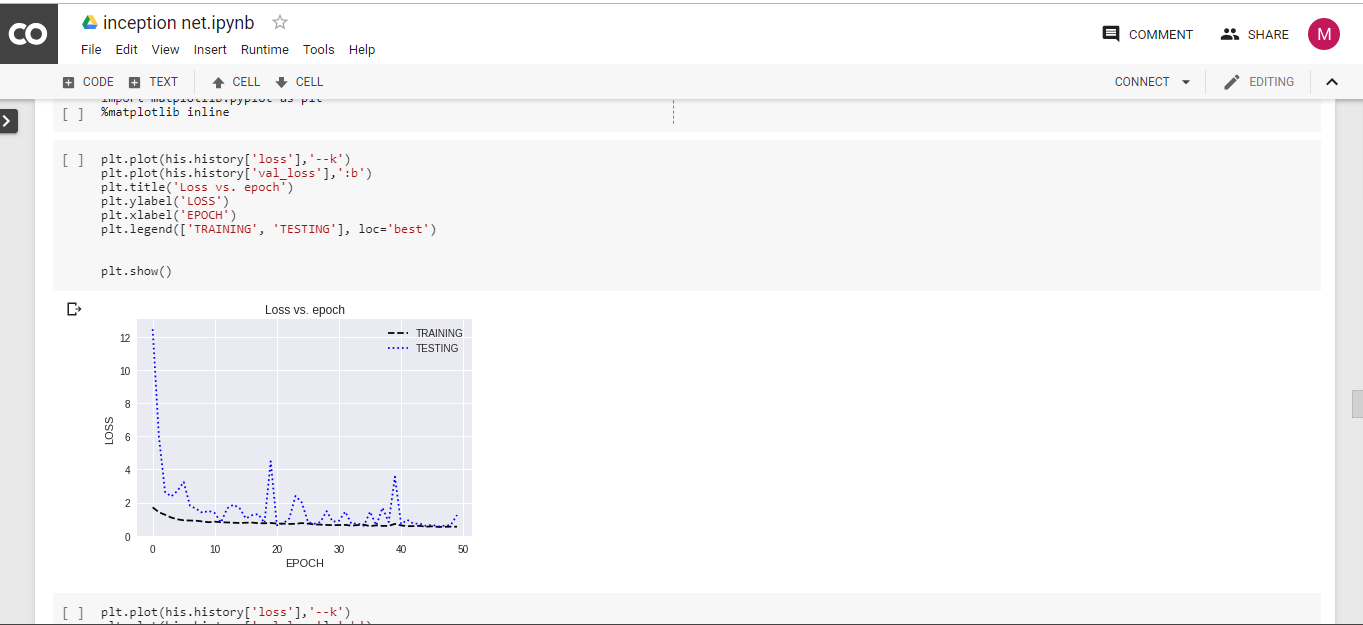


**Fig. 18.** Demonstration of Googlenet



**Fig. 19.** Demonstration of Googlenet

INCEPTION NET:

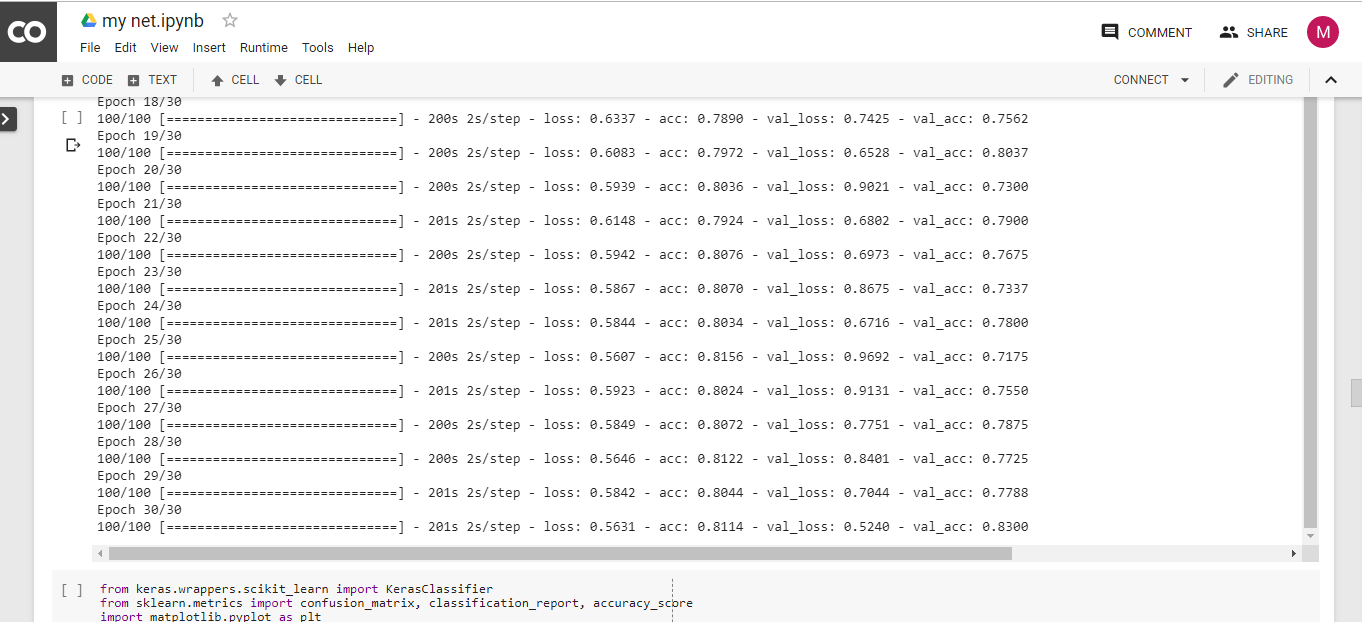


**Fig. 20.** Demonstration of Inception net

PROPOSED MODEL:



**Fig. 21.** Demonstration of proposed model



**Fig. 22.** Demonstration of proposed model

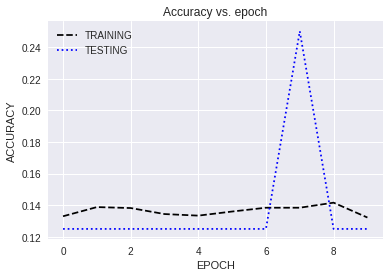
**7. RESULT & DISCUSSION**

VGG16:

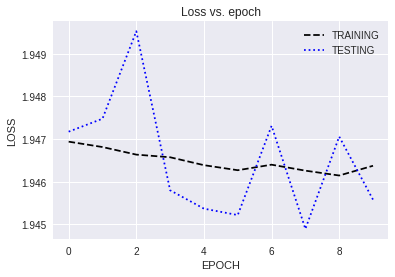
Accuracy :0.142857(14.2857%)

**Table 3.** Classification report of VGG16

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | PRECISION | RECALL | F1-SCORE | SUPPORT |
| 0 | 0 | 0 | 0 | 100 |
| 1 | .14286 | 1 | 0.25 | 100 |
| 2 | 0 | 0 | 0 | 100 |
| 3 | 0 | 0 | 0 | 100 |
| 4 | 0 | 0 | 0 | 100 |
| 5 | 0 | 0 | 0 | 100 |
| 6 | 0 | 0 | 0 | 100 |
| Micro average | 0.14286 | 0.14286 | 0.14286 | 700 |
| Macro average | 0.02041 | 0.14286 | 0.03571 | 700 |
| Weighed average | 0.02041 | 0.14286 | 0.03571 | 700 |



**Fig. 23.** Accuracy of VGG16



**Fig. 24.** Loss of VGG16

VGG 19:

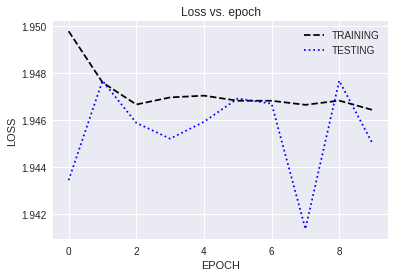
Accuracy :0.142857(14.2857%)

**Table 4.** Classification report of VGG17

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | PRECISION | RECALL | F1-SCORE | SUPPORT |
| 0 | 0.14286 | 1 | 0.25 | 100 |
| 1 | 0 | 0 | 0 | 100 |
| 2 | 0 | 0 | 0 | 100 |
| 3 | 0 | 0 | 0 | 100 |
| 4 | 0 | 0 | 0 | 100 |
| 5 | 0 | 0 | 0 | 100 |
| 6 | 0 | 0 | 0 | 100 |
| Micro average | 0.14286 | 0.14286 | 0.14286 | 700 |
| Macro average | 0.02041 | 0.14286 | 0.03571 | 700 |
| Weighed average | 0.02041 | 0.14286 | 0.03571 | 700 |



**Fig. 25.** Accuracy of VGG19



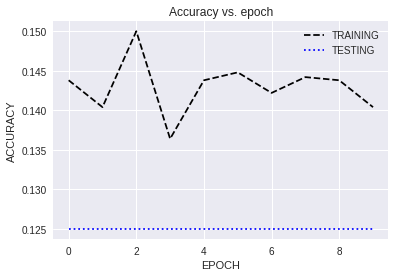
**Fig. 26.** Loss of VGG19

ALEXNET:

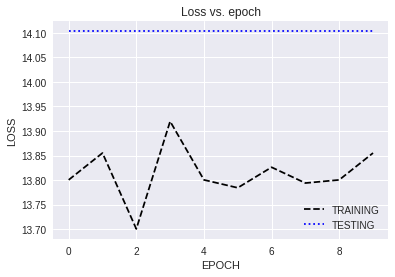
Accuracy :0.142857(14.2857%)

**Table 5.** Classification report of Alexnet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | PRECISION | RECALL | F1-SCORE | SUPPORT |
| 0 | 0 | 0 | 0 | 100 |
| 1 | 0 | 0 | 0 | 100 |
| 2 | 0 | 0 | 0 | 100 |
| 3 | 0 | 0 | 0 | 100 |
| 4 | 0 | 0 | 0 | 100 |
| 5 | 0.14286 | 1 | 0.25 | 100 |
| 6 | 0 | 0 | 0 | 100 |
| Micro average | 0.14286 | 0.14286 | 0.14286 | 700 |
| Macro average | 0.02041 | 0.14286 | 0.03571 | 700 |
| Weighed average | 0.02041 | 0.14286 | 0.03571 | 700 |



**Fig. 27.** Accuracy of Alexnet



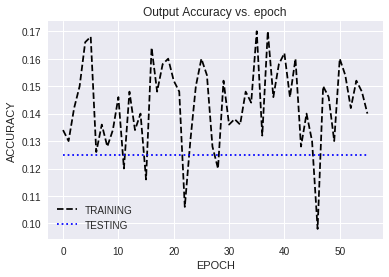
**Fig. 28.** Loss of Alexnet

GOOGLENET:

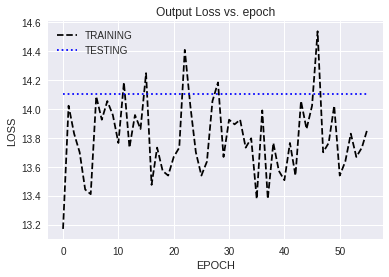
Accuracy :0.142857(14.2857%)

**Table 6.** Classification report of Googlenet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | PRECISION | RECALL | F1-SCORE | SUPPORT |
| 0 | 0 | 0 | 0 | 100 |
| 1 | .0 | 0 | 0 | 100 |
| 2 | .14286 | 1 | 0.25 | 100 |
| 3 | 0 | 0 | 0 | 100 |
| 4 | 0 | 0 | 0 | 100 |
| 5 | 0 | 0 | 0 | 100 |
| 6 | 0 | 0 | 0 | 100 |
| Micro average | 0.14286 | 0.14286 | 0.14286 | 700 |
| Macro average | 0.02041 | 0.14286 | 0.03571 | 700 |
| Weighed average | 0.02041 | 0.14286 | 0.03571 | 700 |



**Fig. 29.** Accuracy of Googlenet



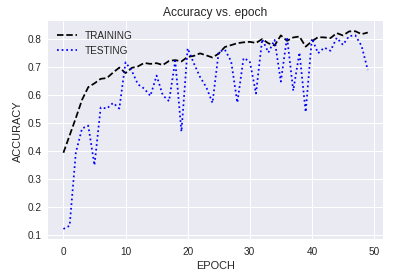
**Fig. 30.** Loss of Googlenet

INCEPTION NET:

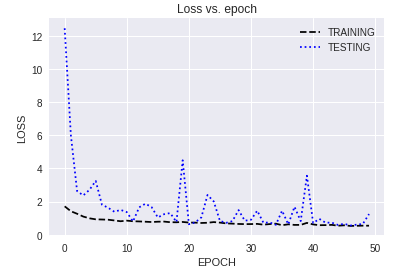
Accuracy :0.664285(6.4285%)

**Table 7.** Classification report of Inception net

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | PRECISION | RECALL | F1-SCORE | SUPPORT |
| 0 | 0.73636 | 0.81 | .77153 | 100 |
| 1 | 0.69231 | 0.45000 | 0.54545 | 100 |
| 2 | 0.90385 | 0.47000 | 0.61842 | 100 |
| 3 | 0.68224 | 0.73 | 0.70531 | 100 |
| 4 | 0.91429 | 0.32 | 0.7407 | 100 |
| 5 | 0.51503 | 0.97 | 0.66897 | 100 |
| 6 | 0.63830 | 0.9 | 0.74689 | 100 |
| Micro average | 0.66429 | 0.66429 | 0.66429 | 700 |
| Macro average | 0.72541 | 0.66429 | 0.64722 | 700 |
| Weighed average | 0.72541 | 0.66429 | 0.64722 | 700 |



**Fig. 31.** Accuracy of Inception net



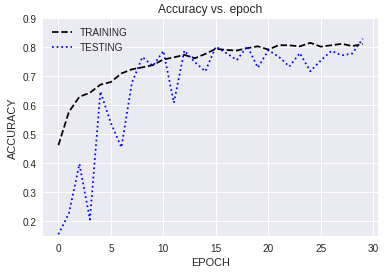
**Fig. 32.** Loss of Inception net

PROPOSED MODEL:

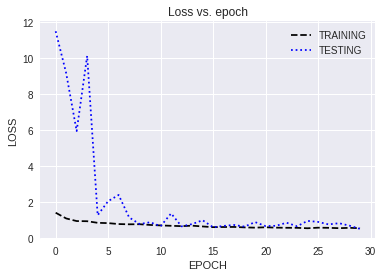
Accuracy : 0.824285(82.4285%)

**Table 8.** Classification report of proposed model

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | PRECISION | RECALL | F1-SCORE | SUPPORT |
| 0 | 0.83333 | 0.80000 | 0.81633 | 100 |
| 1 | 0.81982 | 0.91000 | 0.86256 | 100 |
| 2 | 0.89655 | 0.78000 | 0.83422 | 100 |
| 3 | 0.68644 | 0.81000 | 0.74312 | 100 |
| 4 | 0.96774 | 0.60000 | 0.74074 | 100 |
| 5 | 0.90385 | 0.94000 | 0.92157 | 100 |
| 6 | 0.76230 | 0.93000 | 0.83784 | 100 |
| Micro average | 0.82429 | 0.82429 | 0.82429 | 700 |
| Macro average | 0.83858 | 0.82429 | 0.82234 | 700 |
| Weighed average | 0.83858 | 0.82429 | 0.82234 | 700 |



**Fig. 33.** Accuracy of proposed model



**Fig. 34.** Loss of proposed model

**8. SUMMARY**

The proposed model clearly classifies better than the traditional model. The main reason being the number of parameters in the traditional models are very big in proportion compared to the training data size (5600 images). Thus the model does not train well and acts as a random classifier. One more observation is that the precision of class three is less and the recall of class four is also less. This means that the class three images are misclassified as class four . Thus the CNN does not classify these two classes accurately.

This problem can be overcome if we process classthree separately using image processing techniques. The class three images are straight lines so using Hough transforms, we can setect easily whether a straight line is present or not. So we can train the CNN with the rest of the 6 classes.

## References

[1] SETI <https://www.seti.org/>

[2] Towhid, M.S. and Rahman, M.M., 2017, December. Spectrogram segmentation for bird species classification based on temporal continuity. In *2017 20th International Conference of Computer and Information Technology (ICCIT)*(pp. 1-4). IEEE.

[3] Kalantarian, H., Alshurafa, N., Pourhomayoun, M., Sarin, S., Le, T. and Sarrafzadeh, M., 2014, October. Spectrogram-based audio classification of nutrition intake. In *2014 IEEE Healthcare Innovation Conference (HIC)* (pp. 161-164). IEEE.

[4] Chou, C.H. and Liao, B.J., 2014, April. Music genre classification by analyzing the subband spectrogram. In *2014 International Conference on Information Science, Electronics and Electrical Engineering* (Vol. 3, pp. 1677-1680). IEEE.

[5] Noh, B., Cha, K. and Chang, S., 2017, May. Movement Classification based on Acceleration Spectrogram with Dynamic Time Warping Method. In *2017 18th IEEE International Conference on Mobile Data Management (MDM)*(pp. 397-400). IEEE.

[6] bin Sha'ameri, A.Z. and Lynn, T.J., 2007, May. Spectrogram time-frequency analysis and classification of digital modulation signals. In *2007 IEEE International Conference on Telecommunications and Malaysia International Conference on Communications* (pp. 113-118). IEEE.

[7] Lin, C. and Wang, D., 2010, July. Spectrogram image encoding based on dynamic Hilbert curve routing. In *2010 2nd International Conference on Image Processing Theory, Tools and Applications* (pp. 107-111). IEEE.

[8] Chittora, A. and Patil, H.A., 2014, October. Classification of phonemes using modulation spectrogram based features for Gujarati language. In *2014 International Conference on Asian Language Processing (IALP)* (pp. 46-49). IEEE.

[9] Mustafa, M., Taib, M.N., Murat, Z.H. and Hamid, N.H.A., 2010, November. GLCM texture classification for EEG spectrogram image. In *2010 IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES)* (pp. 373-376). IEEE.

[10] Hansson-Sandsten, M., 2015, August. Classification of bird song syllables using singular vectors of the multitaper spectrogram. In *2015 23rd European Signal Processing Conference (EUSIPCO)* (pp. 554-558). IEEE.

[11] Neammalai, P., Phimoltares, S. and Lursinsap, C., 2014, December. Speech and music classification using hybrid form of spectrogram and fourier transformation. In *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific* (pp. 1-6). IEEE.

[12] Plenary talk at IBM Research Africa.

[13] Harp, G.R. and Rankawat, M., 2018, May. Deep Convolutional Neural Networks for the Generation of High Fidelity Images from Radio Interferometer Visibility Data. In *2018 2nd URSI Atlantic Radio Science Meeting (AT-RASC)* (pp. 1-1). IEEE.