**DOCUMENTATION**

PRESENTED BY :-

SARTHAK BEHERA | SIBASISH SUBHADARSHEE | SUBHRANSU SEKHAR DALAI

CONTENTS

1. USER STORY………………………………………………………………………………………………………….4
2. CURRENT SCENARIO……………………………………………………………………………………………..4
3. FUTURE AIM………………………………………………………………………………………………………….5
4. OUR WORK……………………………………………………………………………………………………………6
   1. TESSERACT ENGIINE…………………………………………………………………………………..7
      1. Theory
      2. Result Analysis
   2. GOOGLE VISION API…………………………………………………………………………………..9
      1. Theory
      2. Code
      3. Implementaion
      4. Result
   3. DEEP LEARNING…………………………………………………………………………………………16
      1. Theory
      2. Code
      3. Implementaion
      4. Result
5. WORK TIMELINE……………………………………………………………………………………………………24
6. SOFTWARES USED…………………………………………………………………………………………………25

**PURPOSE OF THIS DOCUMENT**

This represents an overall thesis on how we can analyse the image present in a given mathematical textbook and represent it simple way so that the visually impaired pupils can understand it.

**SCOPE OF THIS DOCUMENT**

The work in this document is limited only to detection and analysis of mathematical images such as Venn diagrams, geometrical graphs, charts and functions.

**USER STORY:**

As a student with visual disability, I want to read all books available in library.

**CURRENT SCENARIO**

It is believed that the fundamental right to education will bring more pupils with SEN(Special education needs) into ordinary schools, and that this will provide the impetus for change. As stated this will regime a number of innovations in teaching–learning processes, and will also provide pupils with SEN access to a full curriculum in appropriate ways. To facilitate this access, it is important to provide information in Braille, on tape, through sign language, and in simple and straightforward language. Access to the content of the curriculum is further highlighted later in this paper.

Education must aim at developing a system by which abstract concepts are effectively communicated to children with varying learning styles, including those using sign language, Braille, etc

For education using braille,tactile graphs special hardware are being produced for the sole purpose. For every new book published there needs to be a special braille version for SEN with visual needs. So these methods have limitation considering the availability of books produced.

**FUTURE AIM**

To make it possible to read all available books using a technology without producing separate entities speciallyfor visually disabled students.

A system integrated with OCR software to collect all text and tts engine to read out the text to the visually impaired student.

This system will be able to read the exact text with >95%. In case it encounters an image in the book, it will describe the image and its content to the reader.

Using this system visually impaired students will read all available books in the library.

**OUR WORK**

**GOOGLE TESSERACT ENGINE**

The Tesseract Engine is basically an **OCR engine**. The lead developer is Ray Smith. The maintainer is ZdenkoPodobny. Tesseract has **unicode (UTF-8) support**, and can **recognize more than 100 languages** "out of the box". Tesseract supports **various output formats**: plain-text, hocr(html), pdf, tsv, invisible-text-only pdf. You should note that in many cases, in order to get better OCR results, you'll need to **improve the quality of the image** you are giving Tesseract. Tesseract **can be trained to recognize other languages**.

Tesseract was originally developed at Hewlett-Packard Laboratories Bristol and at Hewlett-Packard Co, Greeley Colorado between 1985 and 1994, with some more changes made in 1996 to port to Windows, and some C++izing in 1998. In 2005 Tesseract was open sourced by HP. Since 2006 it is developed by GoogleInc..

**THEORY**

Tesseract was in the top three OCR engines in terms of character accuracy in 1995.It is available for LINUX, WINDOWS and MAC OS X. However, due to limited resources it is only rigorously tested by developers under Windows and UBUNTU.

The first step is a connected component analysis in which outlines of the components are stored. This is a computationally expensive design decision at the time, but has a significant advantage: by inspection of the nesting of outlines, and the number of child and grandchild outlines, it is simple to detect inverse text and recognize it as easily as black-on-white text.Tesseract is probably the first OCR engine able to handle white-on-black text so trivially. At this stage, outlines are gathered together, purely by nesting, into Blobs. Blobs are organized into text lines, and the lines and regions are analysed for fixed pitch or proportional text. Text lines are broken into words differently according to the kind of character spacing. Fixed pitch text is chopped immediately by character cells. Proportional text is broken into words using definite spaces and fuzzy spaces.

Recognition then proceeds as a two-pass process. In the first pass, an attempt is made to recognize each word in turn. Each word that is satisfactory is passed to an adaptive classifier as training data. The adaptive classifier then gets a chance to more accurately recognize text lower down the page. Since the adaptive classifier may have learned something useful too late to make a contribution near the top of the page, a second pass is run over the page, in which words that were not recognized well enough are recognized again.

A final phase resolves fuzzy spaces, and checks alternative hypotheses for the x-height to locate small-cap text.

**CODE**

This is open source code which is available on github.

Link- <https://github.com/tesseract-ocr/tesseract.git>

**RESULTS**

After a thorough research it was found out that the % of accuracy provided by the engine was actually an average value which was nearly about 87%-88%.

The solution in order to improve the accuracy can be achieved in 2 ways.-

1. Pre-processing of the images

The accuracy can be increased if the noises and unnecessary edges are removed from the image before giving to OCR for scan.After the scan is complete, the OCR gives a better outcome.

2. Increasing the trained data

More the data in the dataset better is the accuracy of OCR.

**GOOGLE VISION CLOUD**

Cloud Vision API allows developers to easily integrate vision detection features within applications, including image labelling, face and landmark detection, optical character recognition (OCR), and tagging of explicit content.

**THEORY**

Google Cloud Vision API enables developers to understand the content of an image by encapsulating powerful machine learning models in an easy to use REST API. It quickly classifies images into thousands of categories (e.g., "sailboat", "lion", "Eiffel Tower"), detects individual objects and faces within images, and finds and reads printed words contained within images. You can build metadata on your image catalog, moderate offensive content, or enable new marketing scenarios through image sentiment analysis. Analyse images uploaded in the request or integrate with your image storage on Google Cloud Storage.

Features:-

1. Insight From Your Images

Easily detect broad sets of objects in your images, from flowers, animals, or transportation to thousands of other object categories commonly found within images. Vision API improves over time as new concepts are introduced and accuracy is improved.

2. Power of the Web

Vision API uses the power of Google Image Search to find topical entities like celebrities, logos, or news events. Combine this with Visually Similar Search to find similar images on the web.

**CODE**

#!/usr/bin/python

# -\*- coding: utf-8 -\*-

def run\_quickstart():

# [START vision\_quickstart]

shapes = [

'triangle',

'rectangle',

'quadrilateral',

'square',

'rhombus',

'ellipse',

'oval',

'circle',

'polygon',

'convex polygon',

'concave polygon',

'pentagon',

'hexagon',

'line',

'point',

'angle',

'slope',

'trapezium',

]

best\_description = [

'venn diagram',

'diagram',

'plot',

'graph',

'chart',

'set',

'set notation',

'function',

'graph of function',

'map',

'relation',

'union',

'intersection',

'vector',

'symmetry',

'piecewise linear function',

'injective function',

'surjective function',

'sine wave',

'cos wave',

'quadratic function',

'linear function'

]

import io

import os

list1 = [

'b3',

'b4',

'b5',

'b6',

'b8',

'b9',

'b10',

]

# Imports the Google Cloud client library

# [START migration\_import]

from google.cloud import vision

from google.cloud.vision import types

# [END migration\_import]

# Instantiates a client

# [START migration\_client]

for i in range(len(list1)):

print 'Showing image analysis for ' + list1[i]

print '---------------------------------------------------------------------'

# [END migration\_client]

client = vision.ImageAnnotatorClient()

# The name of the image file to annotate

name = 'resources/' + list1[i] + '.png'

file\_name = os.path.join(os.path.dirname(\_\_file\_\_), name)

# Loads the image into memory

with io.open(file\_name, 'rb') as image\_file:

content = image\_file.read()

image = types.Image(content=content)

# Performs label detection on the image file

# Performs document detection on the image file

# Performs web entity detection on the image file

probable\_desc = []

web\_detection = client.web\_detection(image=image).web\_detection

if web\_detection.web\_entities:

print '\n{} Web entities found: '.format(len(web\_detection.web\_entities))

max\_score = 0

best\_desc = ''

found\_shapes = []

for entity in web\_detection.web\_entities:

for items in best\_description:

if items.lower() == entity.description.lower():

probable\_desc.append(entity.description.lower())

for entity in web\_detection.web\_entities:

for items in best\_description:

if items.lower() == entity.description.lower() and max\_score < float(entity.score):

print 1

max\_score = float(entity.score)

best\_desc = entity.description

if best\_desc.lower() == 'function':

best\_desc = 'graph of function'

for items in shapes:

if items.lower() == entity.description:

found\_shapes.append(items.lower())

print 'Score : {}'.format(entity.score)

print 'Description: {}'.format(entity.description)

response = client.label\_detection(image=image)

labels = response.label\_annotations

print 'Labels:'

flag = 0

for label in labels:

for items in shapes:

if items.lower() == label.description:

found\_shapes.append(items.lower())

print label.description

if best\_desc == '':

for label in labels:

for items in best\_description:

if items == label.description:

best\_desc = label.description

if best\_desc.lower() == 'function':

best\_desc = 'graph of function'

flag = 1

break

if flag == 1:

break

print found\_shapes

print probable\_desc

if(best\_desc != ''):

print 'The image can be best described as a ' + best\_desc.lower()

shape\_string = ''

if(len(found\_shapes) > 1):

shape\_string = 'The shapes found in this image are '

for i in range(len(found\_shapes) - 1):

if(i != len(found\_shapes)-2):

shape\_string = shape\_string + found\_shapes[i] + ', '

else:

shape\_string = shape\_string + found\_shapes[i] + ' '

shape\_string = shape\_string + 'and ' + found\_shapes[len(found\_shapes)-1] +'.'

if(len(found\_shapes) == 1):

shape\_string = 'The shape found in this image is '

shape\_string = shape\_string + found\_shapes[len(found\_shapes)-1] +'.'

print shape\_string

if(best\_desc.lower() == 'plot'):

print 'The ' + best\_desc + ' consists of a x-axis and y-axis are perpendicular intersecting at origin O'

desc\_string = ''

if(len(probable\_desc) > 1):

desc\_string = 'The image may also be described as '

for i in range(len(probable\_desc) - 1):

if(i != len(probable\_desc)-2):

desc\_string = desc\_string + probable\_desc[i] + ' or '

else:

desc\_string = desc\_string + probable\_desc[i] + ' '

desc\_string = desc\_string + 'or ' + probable\_desc[len(probable\_desc)-1] +'.'

if(len(probable\_desc) == 1):

probable\_desc = 'The image may also be described as '

desc\_string = desc\_string + probable\_desc[len(probable\_desc)-1] +'.'

print desc\_string

print '---------------------------------------------------------------------'

print '---------------------------------------------------------------------'

print '---------------------------------------------------------------------'

# perform text detection on image file

# [END vision\_quickstart]

if \_\_name\_\_ == '\_\_main\_\_':

run\_quickstart()

**IMPLEMENTATION**

The vision API was used to detect the mathematical objects and graphs present in the standard 10th mathematics book of Karnataka.

Steps to implement google vision API in local computer:

Step-1: Create a google service account

Step-2: Enable the vision API

1.In the Cloud Platform Console, go to the Manage resources page and select or create a new project.

2. Enable billing of your project.

3. Enable the cloud vision API

Step-3: Download and install and GoogleCloud SDK Command Line tool. Open Google cloud SDK Command Line Tool.

Step-4: Type in the command:

gcloud init

Step-5: Then select the project that was created for vision api purpose and enter in other details like location etc when asked

Step-6: In order to activate the API key for the project type in the command

[gcloud auth application-default login](https://cloud.google.com/sdk/gcloud/reference/beta/auth/application-default/login)

Step-7: Create a folder named resources inside the folder where python code file is present and place the images to be classified inside it. Also change the names of the files in the List list1 in the python code

Step-8: Set the path of the google cloud SDK terminal to where the code file is present and run the code

python FILE\_NAME.py

**RESULT**

We managed to get data using the api but it was not accurate (not as expected).So in the end we came to a conclusion that in order to achieve our solution we need to implement deep learning.

**CONVOLUTIONAL NEURAL NETWORKS FOR IMAGE RECOGNITION**

**INTRODUCTION**

Convolutional neural networks. Sounds like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most influential innovations in the field of computer vision. 2012 was the first year that neural nets grew to prominence as Alex Krizhevsky used them to win that year’s ImageNet competition (basically, the annual Olympics of computer vision), dropping the classification error record from 26% to 15%, an astounding improvement at the time.Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, and Instagram for their search infrastructure. However, the classic, and arguably most popular, use case of these networks is for image processing. Within image processing, let’s take a look at how to use these CNNs for image classification.

**THEORY**

**Inputs and Outputs:-**

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values). Just to drive home the point, let's say we have a color image in JPG form and its size is 480 x 480. The representative array will be 480 x 480 x 3. Each of these numbers is given a value from 0 to 255 which describes the pixel intensity at that point

**What We Want the Computer to Do:-**

Now that we know the problem as well as the inputs and outputs, let’s think about how to approach this. What we want the computer to do is to be able to differentiate between all the images it’s given and figure out the unique features that make a dog a dog or that make a cat a cat. This is the process that goes on in our minds subconsciously as well. When we look at a picture of a dog, we can classify it as such if the picture has identifiable features such as paws or 4 legs. In a similar way, the computer is able perform image classification by looking for low level features such as edges and curves, and then building up to more abstract concepts through a series of convolutional layers. This is a general overview of what a CNN does. Let’s get into the specifics.

**Structure:-**

Back to the specifics. A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, nonlinear, pooling (downsampling), and fully connected layers, and get an output. As we said earlier, the output can be a single class or a probability of classes that best describes the image.

**First Layer – Math Part:-**

The first layer in a CNN is always a Convolutional Layer. First thing to make sure you remember is what the input to this conv (I’ll be using that abbreviation a lot) layer is. Like we mentioned before, the input is a 32 x 32 x 3 array of pixel values. Now, the best way to explain a conv layer is to imagine a flashlight that is shining over the top left of the image. Let’s say that the light this flashlight shines covers a 5 x 5 area. And now, let’s imagine this flashlight sliding across all the areas of the input image. In machine learning terms, this flashlight is called a filter(or sometimes referred to as a neuron or a kernel) and the region that it is shining over is called the receptive field. Now this filter is also an array of numbers (the numbers are called weights or parameters). A very important note is that the depth of this filter has to be the same as the depth of the input (this makes sure that the math works out), so the dimensions of this filter is 5 x 5 x 3. Now, let’s take the first position the filter is in for example.  It would be the top left corner. As the filter is sliding, or convolving, around the input image, it is multiplying the values in the filter with the original pixel values of the image (aka computing element wise multiplications). These multiplications are all summed up (mathematically speaking, this would be 75 multiplications in total). So now you have a single number. Remember, this number is just representative of when the filter is at the top left of the image. Now, we repeat this process for every location on the input volume. (Next step would be moving the filter to the right by 1 unit, then right again by 1, and so on). Every unique location on the input volume produces a number. After sliding the filter over all the locations, you will find out that what you’re left with is a 28 x 28 x 1 array of numbers, which we call an activation map or feature map. The reason you get a 28 x 28 array is that there are 784 different locations that a 5 x 5 filter can fit on a 32 x 32 input image. These 784 numbers are mapped to a 28 x 28 array.

**Going Deeper Through the Network:-**

Now in a traditional convolutional neural network architecture, there are other layers that are interspersed between these conv layers. I’d strongly encourage those interested to read up on them and understand their function and effects, but in a general sense, they provide nonlinearities and preservation of dimension that help to improve the robustness of the network and control overfitting. A classic CNN architecture would look like this.

https://adeshpande3.github.io/assets/Table.png

The last layer, however, is an important one and one that we will go into later on. Let’s just take a step back and review what we’ve learned so far. We talked about what the filters in the first conv layer are designed to detect. They detect low level features such as edges and curves. As one would imagine, in order to predict whether an image is a type of object, we need the network to be able to recognize higher level features such as hands or paws or ears. So let’s think about what the output of the network is after the first conv layer. It would be a 28 x 28 x 3 volume (assuming we use three 5 x 5 x 3 filters).  When we go through another conv layer, the output of the first conv layer becomes the input of the 2nd conv layer.  Now, this is a little bit harder to visualize. When we were talking about the first layer, the input was just the original image. However, when we’re talking about the 2nd conv layer, the input is the activation map(s) that result from the first layer. So each layer of the input is basically describing the locations in the original image for where certain low level features appear. Now when you apply a set of filters on top of that (pass it through the 2nd conv layer), the output will be activations that represent higher level features. Types of these features could be semicircles (combination of a curve and straight edge) or squares (combination of several straight edges). As you go through the network and go through more conv layers, you get activation maps that represent more and more complex features. By the end of the network, you may have some filters that activate when there is handwriting in the image, filters that activate when they see pink objects, etc.

**Fully Connected Layer:-**

Now that we can detect these high level features, the icing on the cake is attaching a **fully connected layer** to the end of the network. This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from.Basically, a FC layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.

**CODE**

**TRAINING**

fromkeras.preprocessing.image import ImageDataGenerator

fromkeras.models import Sequential

fromkeras.layers import Conv2D, MaxPooling2D

fromkeras.layers import Activation, Dropout, Flatten, Dense

fromkeras import backend as K

import cv2

importnumpy as np

# dimensions of our images.

img\_width, img\_height = 150, 150

train\_data\_dir = 'data/train'

validation\_data\_dir = 'data/validation'

nb\_train\_samples = 90

nb\_validation\_samples = 30

epochs = 50

batch\_size = 10

ifK.image\_data\_format() == 'channels\_first':

input\_shape = (3, img\_width, img\_height)

else:

input\_shape = (img\_width, img\_height, 3)

model = Sequential()

model.add(Conv2D(32, (3, 3), input\_shape=input\_shape))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(64))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(5, activation='softmax'))

#model.add(Activation('sigmoid'))

model.compile(loss='categorical\_crossentropy',

optimizer='rmsprop',

metrics=['accuracy'])

# this is the augmentation configuration we will use for training

train\_datagen = ImageDataGenerator(

rescale=1. / 255,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True)

# this is the augmentation configuration we will use for testing:

# only rescaling

test\_datagen = ImageDataGenerator(rescale=1. / 255)

train\_generator = train\_datagen.flow\_from\_directory(

train\_data\_dir,

target\_size=(img\_width, img\_height),

batch\_size=batch\_size,

class\_mode='categorical')

#print (class\_indices)

validation\_generator = test\_datagen.flow\_from\_directory(

validation\_data\_dir,

target\_size=(img\_width, img\_height),

batch\_size=batch\_size,

class\_mode='categorical')

print (validation\_generator.class\_indices)

model.fit\_generator(

train\_generator,

steps\_per\_epoch=nb\_train\_samples // batch\_size,

epochs=epochs,

validation\_data=validation\_generator,

validation\_steps=nb\_validation\_samples // batch\_size)

img = cv2.imread('map\_12.jpg')

img = cv2.resize(img,(150,150))

img = np.reshape(img,[1,150,150,3])

cla = model.predict\_classes(img)

print (cla[0])

if(cla[0] == 0):

print ('venn diagram')

elif(cla[0] == 1):

print ('chart')

elif(cla[0] == 2):

print ('geometry')

elif(cla[0] == 3):

print ('graph')

elif(cla[0] == 4):

print ('map')

model.save('model.h5')

**TESTING**

#print (class\_indices)

validation\_generator = test\_datagen.flow\_from\_directory(

validation\_data\_dir,

target\_size=(img\_width, img\_height),

batch\_size=batch\_size,

class\_mode='categorical')

print (validation\_generator.class\_indices)

model.fit\_generator(

train\_generator,

steps\_per\_epoch=nb\_train\_samples // batch\_size,

epochs=epochs,

validation\_data=validation\_generator,

validation\_steps=nb\_validation\_samples // batch\_size)

# Set the path of the image to be classified

img = cv2.imread('map\_12.jpg')

img = cv2.resize(img,(150,150))

img = np.reshape(img,[1,150,150,3])

cla = model.predict\_classes(img)

print (cla[0])

if(cla[0] == 0):

print ('venn diagram')

elif(cla[0] == 1):

print ('chart')

elif(cla[0] == 2):

print ('geometry')

elif(cla[0] == 3):

print ('graph')

elif(cla[0] == 4):

print ('map')

model.save('model.h5')

**IMPLEMENTATION:**

For implementation of Convolutional Neural Networks we are using a high level deep learning library called Keras with TensorFlowbackend(with GPU support) to train the model for image recognition. We have used Anaconda to create a virtual environment to run our python code.

Steps to set up setting up the virtual environment and Keras with TensorFlow(with GPU support) installation:

**Step 1: Install Anaconda (Python 3.6 version)**

**Step 2: Update Anaconda**

**Open Anaconda Prompt to type the following command(s)**

conda update conda

conda update --all

**Step 3: Install CUDA Tookit 8.0**

Choose your version depending on your Operating System

**Step 4: Download cuDNN**[**Download**](https://developer.nvidia.com/rdp/cudnn-download)

Choose your version depending on your Operating System. Membership registration is required.

Put your unzipped folder in C drive as follows:

C:\cudnn-8.0-windows10-x64-v5.1

**Step 5: Add cuDNN into Environment PATH**[**Tutorial**](https://kb.wisc.edu/cae/page.php?id=24500)

Add the following path in your Environment. Subjected to changes in your installation path.

C:\cudnn-8.0-windows10-x64-v5.1\cuda\bin

Turn off all the prompts. Open a new Anaconda Prompt to type the following command(s)

echo %PATH%

You shall see that the new Environment PATH is there.

**Step 6: Create an Anaconda environment with Python=3.5**

Open Anaconda Prompt to type the following command(s)

conda create -n tensorflow python=3.5 numpy scipy matplotlib spyder

**Step 7: Activate the environment**

Open Anaconda Prompt to type the following command

activate tensorflow

**Step 8: Install TensorFlow-GPU-1.0.1**

Open Anaconda Prompt to type the following command(s)

pip3 install --upgrade tensorflow-gpu

**Step 9: Install Keras**

Open Anaconda Prompt to type the following command(s)

pip install keras

pip install pillow

pip install h5.py

conda install -c conda-forge opencv

After keras installation in order to implement the code we have to create the dataset for training as well as testing the model. Steps to create the dataset:

**Step 1**: Create a folder named data inside the folder where python code file is present . Within the folder create 2 subfolders named train and validation

**Step 2**: Within each of these 2 subfolders train and validation create subfolders and name these subfolders according to the classes based on which we want to classify the images.

**Step 3**: Then accordingly store the images in the subfolders. Store more images in the train subfolder.

**Step 4**: To train the CNN model run the training python code given above in code section

python TRAINING\_CODE.py

**Step 5**: After training run the testing code given above in code section to test the CNN model

python TESTING\_CODE.py

**Architecture:**

data

train

chart

venn diagram

gometry

graph

map

validation

chart

venn diagram

geometry

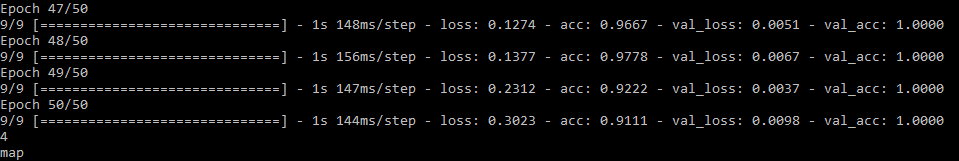
graph

map

**Step-4**: Then set the path of anaconda prompt to where the code file is located and run the command

python FILE\_NAME.py

**RESULT:**

The convolutional neural network model was trained with the training data set for 50 epochs. After each epoch the accuracy as compared with the validation data set increased as the model learnt to classify the images according to the respective classes. After 50 epochs the validation accuracy came out to be 1.0000 and the training data set accuracy came out to be 0.9111.

**WORK TIMELINE**

|  |  |  |  |
| --- | --- | --- | --- |
| **STAGES** | **START DATE** | **END DATE** | **DURATION** |
| Scope of work | 8.12.17 | 8.12.17 | 1 |
| Research on Tesseract Engine | 9.12.17 | 10.12.17 | 2 |
| Documentation of our research | 11.12.17 | 11.12.17 | 1 |
| Introduction to Google Vision api | 12.12.17 | 12.12.17 | 1 |
| Comparision of Vision/AWS/Microsoft azure | 13.12.17 | 14-12-207 | 2 |
| Implementation of Vision API to detect images of the given Math Text Book | 14.12.17 | 15.12.17 | 2 |
| Training of Datasets | 15.12.17 | 16.12.17 | 2 |
| Documenation of result using Vision API | 16.12.17 | 16.12.17 | 1 |
| Introduction to Deep Learning | 17.12.17 | 18.12.17 | 2 |
| Coding Phase I | 18.12.17 | 19.12.17 | 2 |
| Training of Datasets | 19.12.17 | 19.12.17 | 1 |
| Coding Phase II | 20.12.17 | 20.12.17 | 1 |
| Testing Phase | 20.12.17 | 21.12.17 | 2 |
| Result Analysis and Documentation | 22.12.17 | 22.12.17 | 1 |

**SOFTWARES USED**

1. Machine Learning

- TensorFlow(as a backend)

- Keras Library(for Deep learning)

1. Virtual Enviroment

- Anaconda 3.2 (Python 3.5)

1. API

- Google Vision API

1. OCR Engine

-Tesseract Engine