A PROJECT REPORT

on

"MACHINE LEARNING APPLICATION IN STELLAR FIELD IDENTIFICATION"

Submitted to

KIIT Deemed to be University

In Partial Fulfillment of the Requirement for the Award of

BACHELOR'S DEGREE IN INFORMATION TECHNOLOGY

BY

HARSHITA ARYA 1605509

UNDER THE GUIDANCE OF
PROF. MAHENDRA KUMAR GOURISARIA

SCHOOL OF COMPUTER ENGINEERING
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA - 751024

April 2020

A PROJECT REPORT

on

"MACHINE LEARNING APPLICATION IN STELLAR FIELD IDENTIFICATION"

Submitted to

KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR'S DEGREE IN INFORMATION TECHNOLOGY

BY

HARSHITA ARYA 1605509

UNDER THE GUIDANCE OF

PROF. MAHENDRA KUMAR GOURISARIA

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

BHUBANESWAE, ODISHA -751024

April 2020

KIIT Deemed to be University

School of Computer Engineering
Bhubaneswar, ODISHA 751024

CERTIFICATE

This is certify that the project entitled

"MACHINE LEARNING APPLICATION IN STELLAR FIELD IDENTIFICATION"

submitted by

HARSHITA ARYA 1605509

is a record of bonafide work carried out by them, in the partial fulfillment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering OR Information Technology) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2019-2020, under our guidance.

Date:04/04/2020

Prof. Mahendra Kumar Gourisaria
Project Guide

Acknowledgements

We are profoundly grateful to Prof. Mahendra Kumar Gourisaria for his expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion . His regular advices and progress checks helped me learn how to undertake a project and gave me a smooth way to carry out the project and kept me confident in what I was doing.

I will always be thankful to his contributions as a guide and hope to follow his principles in future projects to accomplish more projects.

HARSHITA ARYA

ABSTRACT

This project aims to use Deep Learning concepts to predict the accurate positions of stars.Our earth undergo various situations which cause the inaccuracy in measuring the position of the stars in a particular field. These are all caused due to the celestial movements of both the stars and our earth.Contrary to our widespread belief the stars are moving and are not stationary. They have been a the cause of error since a long time. The causes are Abberation , Parallax , Rotation , Nutation and various other factors .

Our approach is to use Convolutional Neural Network to account for the different causes of accuracy. The model to be used is ALEXNET which is a very renowned model in Image related problem solving. Having won the ImageNet 2012 challege it is worth trusting for our error detection problem.The data was taken from the ipyaladin website which is the most diverse source of dataset in the world of astronomy and the co ordinate system used is RA-Dec system which is again the most widely used locating system.

For the Deep Learning part we have identified the problem as a supervised regression domain of problem and its input would be the images that would be taken as input and the output would be a set of corrected co ordinate system. We attempt to achieve the highest accuracy possible and keep improving our hyper parameters to further improve our model.

Keywords: Deep Learning, Supervised Learning, Convolutional Neural Network, Nutation, Abberartion, Parallax.

Contents

1	Introduction	
	1.1 RA-Dec Co-Ordinate system	1-7
	1.1.1 RA	1-3
	1.1.2 Dec	3-7
2	Literature Survey	8-10
	2.1 Deep Learning	8-10
3	Software Requirements Specification	11
	3.1 List of Softwares	11
	..	
4	Requirement Analysis	12
	4.1 Requirement Analysis	12
5	System Design	6
	5.1 System design	6
6	System Testing	13-14
	6.1 Test Cases and Test Results	13-14
7	Implementation	15-28
8	Screenshots of Project	29-31
	8.1 Convolution Layers	29-31
9	Conclusion and Future Scope	32
	10.1 Conclusion	32
	10.2 Future Scope	32
11	References	33

List of Figures

1.1	RA-Dec System	1
2.1	CDS Data	4
3.1	Convolution Layers	5
4.1	AlexNet	13
5.1	Accuracy and Loss	15
6.1	Convolution weights and biases	29-31

Chapter 1

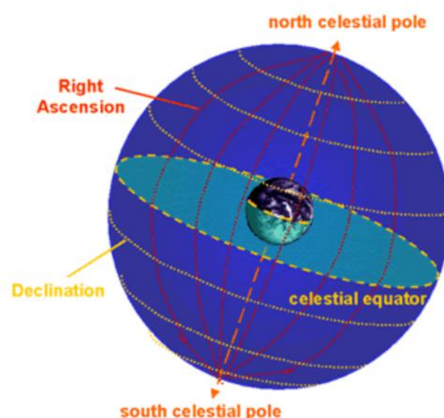
Introduction

1.1 The RA-Dec Co Ordinate System

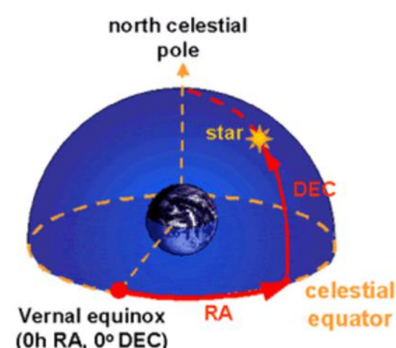
1.1.1 The Need of Celestial Co ordinate system

This is the preferred coordinate system to pinpoint objects on the celestial sphere. Unlike the horizontal coordinate system, equatorial coordinates are independent of the observer's location and the time of the observation. This means that only one set of coordinates is required for each object, and that these same coordinates can be used by observers in different locations and at different times.

The equatorial coordinate system is basically the projection of the latitude and longitude coordinate system we use here on Earth, onto the celestial sphere. By direct analogy, lines of latitude become lines of declination (Dec; measured in degrees, arcminutes and arcseconds) and indicate how far north or south of the celestial equator (defined by projecting the Earth's equator onto the celestial sphere) the object lies. Lines of longitude have their equivalent in lines of right ascension (RA), but whereas longitude is measured in degrees, minutes and seconds east the Greenwich meridian, RA is measured in hours, minutes and seconds east from where the celestial equator intersects the ecliptic (the vernal equinox).



The right ascension coordinate is analogous longitude here on Earth.



The right ascension of an object indicates its angular distance from the Vernal Equinox.

1.1.2 Right Ascension(RA)

Analogous to the longitude coordinate here on Earth, RA is also an angular distance, measured eastward from the vernal equinox (where the celestial equator intersects the ecliptic). However, primarily for historial reasons, RA is measured in hours, minutes and seconds rather than degrees, minutes and seconds, with 1 hour of RA measured at the celestial equator equal to 15 degrees.

1.1.3 Declination (Dec)

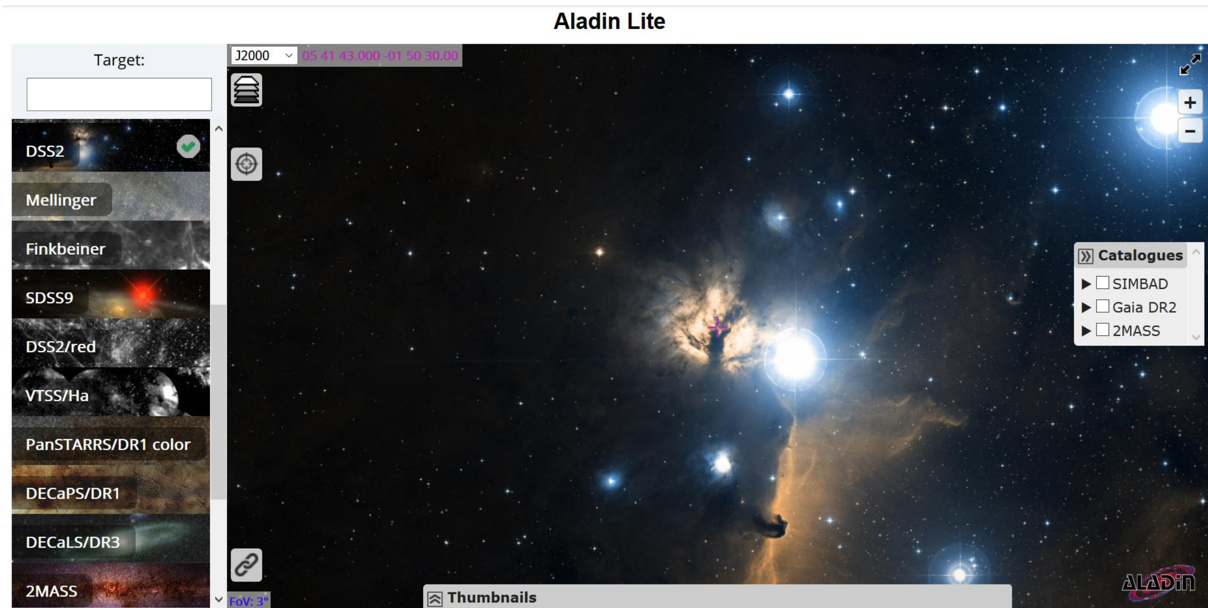
Along with the right ascension (RA) and epoch, the 'declination' (Dec) of an object is used to define its position on the celestial sphere in the equatorial coordinate system.
Measured in degrees, arcminutes and arcseconds it defines how far north (positive Dec) or south (negative Dec) of the celestial equator the object lies, and is directly analogous to the latitude coordinate here on Earth. Stars on the celestial equator have Dec=0°, stars at the south celestial pole have Dec=-90°, and stars at the north celestial pole have Dec=$+90^o$.

At first glance, this system of uniquely positioning an object through two coordinates appears easy to implement and maintain. However, the equatorial coordinate system is tied to the orientation of the Earth in space, and this changes over a period of 26,000 years due to the precession of the Earth's axis. We therefore need to append an additional piece of information to our coordinates – the epoch. For example, the Einstein Cross (2237+0305) was located at RA = 22h 37m, Dec = $+03^o05'$ using epoch B1950.0. However, in epoch J2000.0 coordinates, this object is at RA = 22h 37m, Dec = $+03^o$ 21'. The object itself has not moved – just the coordinate system.
The equatorial coordinate system is alternatively known as the 'RA/Dec coordinate system' after the common abbreviations of the two components involved.

1.2 The CDS Data Source

The equatorial coordinate system is alternatively Strasbourg astronomical Data Center (CDS) is dedicated to the collection and worldwide distribution of astronomical data and related information.

The CDS hosts the SIMBAD astronomical database, the world reference database for the identification of astronomical objects; VizieR, the catalogue service for the CDS reference collection of astronomical catalogues and tables published in academic journals; and the Aladin interactive software sky atlas for access, visualization and analysis of astronomical images, surveys, catalogues, databases and related data.



1.2 Convolutional Neural Network (CNN)

1.2.1 Introduction

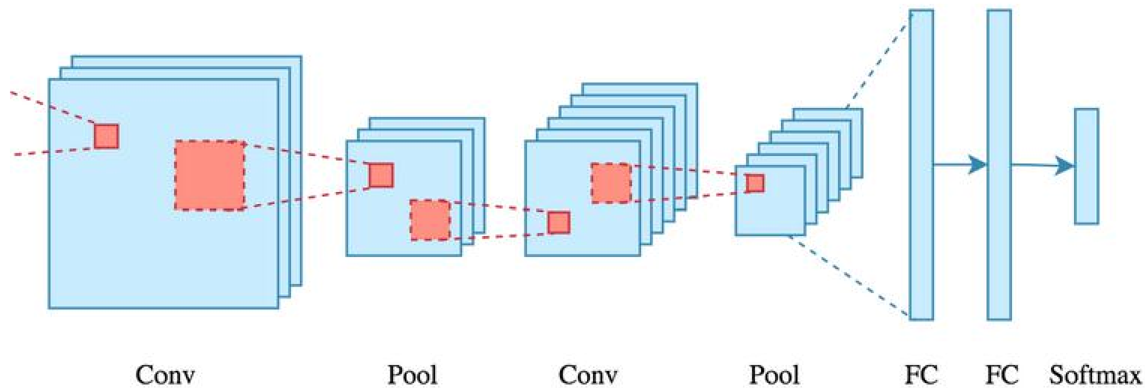
Convolutional Neural Networks (CNN) are everywhere. It is arguably the most popular deep learning architecture. The recent surge of interest in deep learning is due to the immense popularity and effectiveness of convnets. The interest in CNN started with AlexNet in 2012 and it has grown exponentially ever since. In just three years, researchers progressed from 8 layer AlexNet to 152 layer ResNet.

CNN is now the go-to model on every image related problem. In terms of accuracy they blow competition out of the water. It is also successfully applied to recommender systems, natural language processing and more. The main advantage of CNN compared to its predecessors is that it automatically detects the important features without any human supervision. For example, given many pictures of cats and dogs it learns distinctive features for each class by itself.

CNN is also computationally efficient. It uses special convolution and pooling operations and performs parameter sharing. This enables CNN models to run on any device, making them universally attractive.

All in all this sounds like pure magic. We are dealing with a very powerful and efficient model which performs automatic feature extraction to achieve superhuman accuracy (yes CNN models now do image classification better than humans).

1.2.2 Architecture



There is an input image that we're working with. We perform a series convolution + pooling operations, followed by a number of fully connected layers. If we are performing multiclass classification the output is softmax. We will now dive into each component.

Convolution

The main building block of CNN is the convolutional layer. Convolution is a mathematical operation to merge two sets of information. In our case the convolution is applied on the input data using a *convolution filter* to produce a *feature map*. There are a lot of terms being used so let's visualize them one by one.

The convolution operation by sliding this filter over the input. At every location, we do element-wise matrix multiplication and sum the result. This sum goes into the feature map. The green area where the convolution operation takes place is called the *receptive field*. One more important point before we visualize the actual convolution operation. We perform multiple convolutions on an input, each using a different filter and resulting in a distinct feature map. We then stack all these feature maps together and that becomes the final output of the convolution layer. But first let's start simple and visualize a convolution using a single filter..

To help with visualization, we slide the filter over the input as follows. At each location we get a scalar and we collect them in the feature map. The animation shows the sliding operation at 4 locations, but in reality it's performed over the entire input.Below we can see how two feature maps are stacked along the depth dimension. The convolution operation for each filter is performed independently and the resulting feature maps are disjoint.

Non-linearity

For any kind of neural network to be powerful, it needs to contain non-linearity. Both the ANN and autoencoder we saw before achieved this by passing the weighted sum of its inputs through an activation function, and CNN is no different. We again pass the result of the convolution operation through *relu* activation function. So the values in the final feature maps are not actually the sums, but the relu function applied to them. We have omitted this in the figures above for simplicity. But keep in mind that any type of convolution involves a relu operation, without that the network won't achieve its true potential.

Stride and Padding

Stride specifies how much we move the convolution filter at each step. By default the value is 1, as you can see in the figure below.We can have bigger strides if we want less overlap between the receptive fields. This also makes the resulting feature map smaller since we are skipping over potential locations. The following figure demonstrates a stride of 2. Note that the feature map got smaller.We see that the size of the feature map is smaller than the input, because the convolution filter needs to be contained in the input. If we want to maintain the same dimensionality, we can use *padding* to surround the input with zeros.

The gray area around the input is the padding. We either pad with zeros or the values on the edge. Now the dimensionality of the feature map matches the input. Padding is commonly used in CNN to preserve the size of the feature maps, otherwise they would shrink at each layer, which is not desirable. The 3D convolution figures we saw above used padding, that's why the height and width of the feature map was the same as the input (both 32x32), and only the depth changed.

Pooling

After a convolution operation we usually perform *pooling* to reduce the dimensionality. This enables us to reduce the number of parameters, which both shortens the training time and combats overfitting. Pooling layers downsample each feature map independently, reducing the height and width, keeping the depth intact.

The most common type of pooling is *max pooling* which just takes the max value in the pooling window. Contrary to the convolution operation, pooling has no parameters. It slides a window over its input, and simply takes the max value in the window. Similar to a convolution, we specify the window size and stride.

By halving the height and the width, we reduced the number of weights to 1/4 of the input. Considering that we typically deal with millions of weights in CNN architectures,

Fully Connected

After the convolution + pooling layers we add a couple of fully connected layers to wrap up the CNN architecture. .

The output of both convolution and pooling layers are 3D volumes, but a fully connected layer expects a 1D vector of numbers. So we *flatten* the output of the final pooling layer to a vector and that becomes the input to the fully connected layer. Flattening is simply arranging the 3D volume of numbers into a 1D vector,

Chapter 2

Literature Survey

2.1 The different causes of error in the position detection

2.1.1 The Earth's Precession and Nutation

The Earth's axis rotates (precesses) just as a spinning top does. The period of precession is about 26,000 years. Therefore, the North Celestial Pole will not always be point towards the same starfield.Precession is caused by the gravitational pull of the Sun and the Moon on the Earth.

2.1.2 The Proper Motion of Stars

The period of Proper motion, in astronomy, the apparent motion of a star across the celestial sphere at right angles to the observer's line of sight; any radial motion (toward or away from the Sun) is not included. It is observed with respect to a framework of very distant background stars or galaxies. Proper motion is generally measured in seconds of arc per year; the largest known is that of Barnard's star in the constellation Ophiuchus, about 10″ yearly. The English astronomer Edmond Halley, in 1718, was the first to detect proper motions—those of Arcturus and Sirius. The symbol for proper motion is the Greek letter μ (mu).

2.1.3 Julian Day

The **Julian date** (JD) is a continuous count of days and fractions elapsed since the same initial epoch. Currently the JD is 2458941.8326389. The integral part (its floor) gives the Julian day number. The fractional part gives the time of day since noon UT as a decimal fraction of one day or fractional day, with 0.5 representing midnight UT. Typically, a 64-bit floating point (double precision) variable can represent an epoch expressed as a Julian date to about 1 millisecond precision.

A Julian date of 2454115.05486 means that the date and Universal Time is Sunday 14 January 2007 at 13:18:59.9.

The decimal parts of a Julian date:
0.1 = 2.4 hours or 144 minutes or 8640 seconds
0.01 = 0.24 hours or 14.4 minutes or 864 seconds
0.001 = 0.024 hours or 1.44 minutes or 86.4 seconds
0.0001 = 0.0024 hours or 0.144 minutes or 8.64 seconds
0.00001 = 0.00024 hours or 0.0144 minutes or 0.864 seconds.

2.1.4 Sidreal Day

The sidereal day is the time it takes for the Earth to complete one rotation about its axis with respect to the ‘fixed’ stars. By fixed, we mean that we treat the stars as if they were attached to an imaginary celestial sphere at a very large distance from the Earth.A measurement of the sidereal day is made by noting the time at which a particular star passes the celestial meridian (i.e. directly overhead) on two successive nights. On Earth, a sidereal day lasts for 23 hours 56 minutes 4.091 seconds, which is slightly shorter than the solar day measured from noon to noon.Our usual definition of an Earth day is 24 hours, so the sidereal day is 4 minutes faster. This means that a particular star will rise 4 minutes earlier every night, and is the reason why different constellations are only visible at specific times of the year.

2.1.5 Astronomical Time

The kind of Time we deal with is related to the sun and its position in the sky relative to the line called local celestial meridian,that runs from south point in the horizon to the zenith directly overhead to the north point in the horizon to the nadir directly under the foot and back to the south point.

2.1.6 Rotation of the Earth and Revolution around the Sun

There can be a lot of irregularities due to the rotation and the revolution of the sun:

1. The eastward motion of the sun along the ecliptic is not uniform. The earth is closer to the sun in January(Perihelon) and farther away in July(Apihelion) , when it is closer to the sun it moves faster and when it is farther it moves slowly.

2.Another irregularity in the motion of the sun arises because the sun does not follow the celestial equator
If it moves 1° eastward along the celestial equator it's RA would increase by 4 min
However during the time of solstices the sun lies 23 $1/2^o$ N/S of celestial equator, here the lines in ascension is closer together, when the sun moves 1° eastward along the ecliptic , its RA can increase as much as 4.4 min

3.Yet, Another irregularity is due to the inclination of the ecliptic to the celestial equator. When the sun is moving around the solstices it moves around 1º per day eastward.At the time of equinoxes the motion is not parallel to the ceelestial equator.In spring the sun's motion is directed partially towards east and partially towards North as it returns to the Northern Hemisphere.In Spring the sun's motion is tow

Chapter 3

Software Requirements Specification

3.1 The Software Needed for the Implementation of the project are:

3.1.1 Tensorflow 1.0
3.1.2 Windows 10
3.1.3 Python 3
3.1.4 Aladin browser

Chapter 4

Requirement Analysis

4.1 Deep Learning Approach to Error Detection

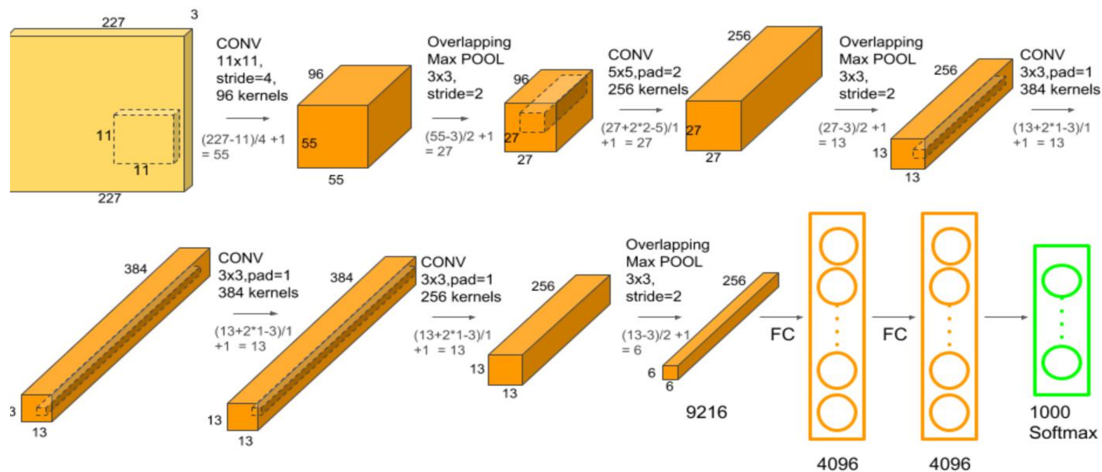
In astronomy, the volume and complexity is increasing all the time, which can be challenging for traditional analysis methods. The rapid progress in machine learning and deep learning techniques offer us an opportunity to approach these problems in different ways To solve the error detection problem, Deep Learning offers a wide range of Image Detection, Classification and various other models for the Analysis.The deep Learning Approach is the most convenient approach to handle such problems.

Thus an image of a part of the field is taken from the Alladin Browser and it is required to find the accurate positioning of the stars using AlenNet Model. The required output is the correction in the dataset needed due to the various factors listed above in the RA-Dec system of co ordinates.

Chapter 5

System Design

5.1 The Design of the AlexNet Model



The since our problem is of nature supervised regression we will have to use only some of the layers of the model.To get the regression data we would have to remove the softmax regression layer.

The image data usually given to the model is of size 28x28 thus an image resizing is require to be performed to be feeded into the channel.

Atleast 50000 data is to be used for the testing and the Validation set from the PanSTARRS data of the Alladin browser.

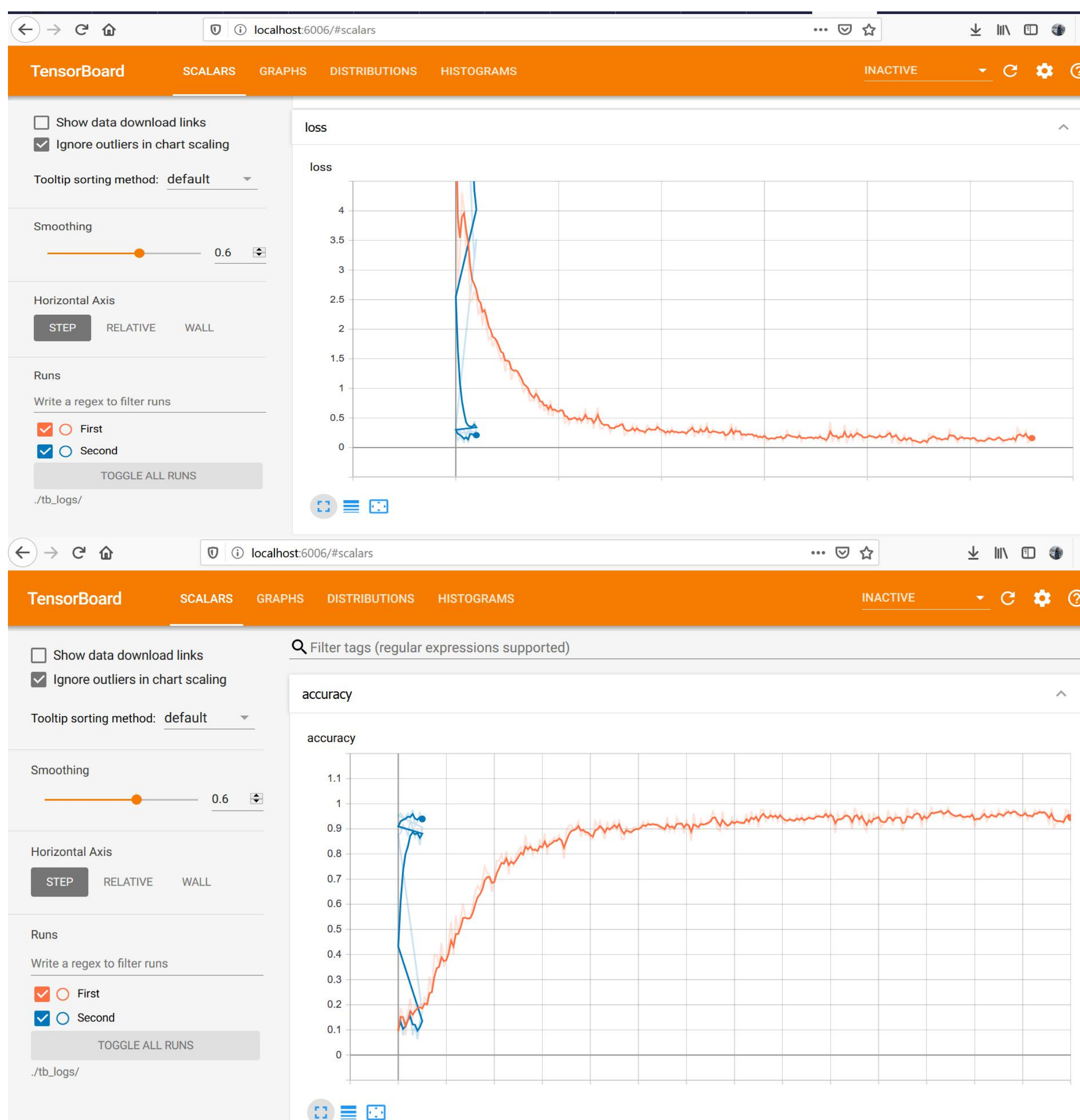
A plot is to be maintained in the tensorboard and the accuray and the various convolution layers are to be analysed , with respect to their weights and biases.

Chapter 6

System Testing

WRITE HERE.

6.1 Test Cases and Test Results



Implementation

7.1 Train.py

```
import os
import numpy as np
import tensorflow.compat.v1 as tf
from model import alexnet
from evals import calc_loss_acc, train_op
from tensorboard.plugins.hparams import api as hp
import input_data
from scipy import misc
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)


def del_all_flags(FLAGS):
    flags_dict = FLAGS._flags()
    keys_list = [keys for keys in flags_dict]
    for keys in keys_list:
        FLAGS.__delattr__(keys)

del_all_flags(tf.flags.FLAGS)

flags = tf.app.flags
FLAGS = flags.FLAGS

tf.app.flags.DEFINE_string('f', '', 'kernel')

flags.DEFINE_integer('valid_steps', 11, 'The number of validation
steps ')
flags.DEFINE_integer('max_steps', 300, 'The number of maximum
steps for traing')
```

```
flags.DEFINE_integer('batch_size', 128, 'The number of images in each batch during training')

flags.DEFINE_float('base_learning_rate', 0.0001, "base learning rate for optimizer")

flags.DEFINE_integer('input_shape', 784 , 'The inputs tensor shape')
##flags.DEFINE_integer('input_shape', 784, 'The inputs tensor shape')

flags.DEFINE_integer('num_classes', 10, 'The number of label classes')

flags.DEFINE_string('save_dir', './outputs', 'The path to saved checkpoints')

flags.DEFINE_float('keep_prob', 0.75, "the probability of keeping neuron unit")

flags.DEFINE_string('tb_path', './tb_logs/First/', 'The path points to tensorboard logs ')

import pandas as pd

dict = {row[0] : row[1] for _, row in pd.read_csv("C:/Users/KIIT/Anaconda3/envs/tensorflow_env/MNIST-AlexNet-Using-Tensorflow-master/MNIST.csv",header=None).iterrows()}

images = []
labels=[]
files=[]
for filename in os.listdir("C:/Users/KIIT/Anaconda3/envs/tensorflow_env/MNIST-
```

```
AlexNet-Using-Tensorflow-master/data/mnistasjpg/trainingSet"):
  img =
cv.imread(os.path.join("C:/Users/KIIT/Anaconda3/envs/tensorflow_
env/MNIST-AlexNet-Using-Tensorflow-
master/data/mnistasjpg/trainingSet", filename))
  if img is not None:
      files.append(filename)
      images.append(img)
      labels.append(dict.get(filename))

temp = list(zip(images, labels))
random.shuffle(temp)
images, labels = zip(*temp)

imgf=np.array(images)
imagef=tf.image.rgb_to_grayscale(imgf)

"""
batch = mnist.train.next_batch(55000)
X_batch = batch[0]
batch_tensor = tf.reshape(X_batch, [55000, 28, 28, 1])
resized_images = tf.image.resize_images(batch_tensor, [25,25])
"""
numpy_imgs = imagef.numpy()
im=tf.reshape(numpy_imgs, [42000,784])
imf=im.numpy()
def train(FLAGS):
    """Training model

    """
    valid_steps = FLAGS.valid_steps
    max_steps = FLAGS.max_steps
    batch_size = FLAGS.batch_size
    base_learning_rate = FLAGS.base_learning_rate
    input_shape = FLAGS.input_shape  # image shape = 28 * 28
```

```
num_classes = FLAGS.num_classes
  keep_prob = FLAGS.keep_prob
  save_dir = FLAGS.save_dir
  tb_path = FLAGS.tb_path

  train_loss, train_acc = [], []
  valid_loss, valid_acc = [], []

  tf.reset_default_graph()
  # define default tensor graphe
  with tf.Graph().as_default():
      images_pl = tf.placeholder(tf.float32, shape=[None,
input_shape])
      labels_pl = tf.placeholder(tf.float32, shape=[None,
num_classes])

      # define a variable global_steps
      global_steps = tf.Variable(0, trainable=False)

      # build a graph that calculate the logits prediction from model
      logits = alexnet(images_pl, num_classes, keep_prob)

      loss, acc, _ = calc_loss_acc(labels_pl, logits)

      # build a graph that trains the model with one batch of example
and updates the model params
      training_op = train_op(loss, global_steps, base_learning_rate)
      validing_op = train_op(loss, global_steps, base_learning_rate)
      # define the model saver
      saver = tf.train.Saver(tf.global_variables())

      # define a summary operation
      summary_op = tf.summary.merge_all()
      summ_op=tf.summary.merge_all()
```

```
with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        # print(sess.run(tf.trainable_variables()))
        # start queue runners
        coord = tf.train.Coordinator()
        threads = tf.train.start_queue_runners(sess=sess,
coord=coord)
        train_writter = tf.summary.FileWriter(tb_path, sess.graph)
        train_writter2 = tf.summary.FileWriter('./tb_logs/Second/',
sess.graph)
        #train_writter = tf.summary.create_file_writer(tb_path)

        # start training
        for step in range(max_steps):

            train_image_batch, train_label_batch =
mnist.train.next_batch(batch_size)
            train_feed_dict = {images_pl: train_image_batch,
labels_pl: train_label_batch}

            _, _loss, _acc, _summary_op = sess.run([training_op, loss,
acc, summary_op], feed_dict = train_feed_dict)

            # store loss and accuracy value
            train_loss.append(_loss)
            train_acc.append(_acc)
            print("Iteration " + str(step) + ", Mini-batch Loss= " +
"{:.6f}".format(_loss) + ", Training Accuracy= " +
"{:.5f}".format(_acc))
            train_writter.add_summary(_summary_op, global_step=
step)
            print("brrr",step)
            if step % 100 == 0:
                _valid_loss, _valid_acc = [], []
                print('Start validation process')
```

```
for itr in range(valid_steps):
                    valid_image_batch, valid_label_batch =
mnist.test.next_batch(batch_size)

                    valid_feed_dict = {images_pl: valid_image_batch,
labels_pl: valid_label_batch}

                    _,_loss, _acc,_validing_op =
sess.run([validing_op,loss, acc,summ_op], feed_dict =
valid_feed_dict)
                    train_writter2.add_summary(_validing_op,
global_step= itr)

                    _valid_loss.append(_loss)
                    _valid_acc.append(_acc)
                    #train_writter.add_summary(_summary_op,
global_step= step)

                valid_loss.append(np.mean(_valid_loss))
                valid_acc.append(np.mean(_valid_acc))

                #print("Iteration {}: Train Loss {:6.3f}, Train Acc
{:6.3f}, Val Loss {:6.3f}, Val Acc {:6.3f}".format(itr, train_loss[-1],
train_acc[-1], valid_loss[-1], valid_acc[-1]))
                print("Iteration {}: Train Loss {}, Train Acc {}, Val
Loss {}, Val Acc {}",itr, train_loss[-1], train_acc[-1], valid_loss[-1],
valid_acc[-1])
                        #train_writter.add_summary(_summary_op,
global_step= step)
                #print("brrr",step)
                #train_writter.
                #with train_writter.as_default():
                #tf.summary.scalar('loss', _valid_loss)
                #tf.summary.scalar('accuracy', _valid_acc)
np.save(os.path.join(save_dir, 'accuracy_loss', 'train_loss'),
train_loss)
```

```
  np.save(os.path.join(save_dir, 'accuracy_loss', 'train_acc'),
train_acc)
            np.save(os.path.join(save_dir, 'accuracy_loss', 'valid_loss'),
valid_loss)
            np.save(os.path.join(save_dir, 'accuracy_loss', 'valid_acc'),
valid_acc)
            checkpoint_path = os.path.join(save_dir, 'model',
'model.ckpt')
            saver.save(sess, checkpoint_path, global_step=step)

            coord.request_stop()
            coord.join(threads)
    sess.close()
if __name__ == '__main__':
    train(FLAGS)
```

7.2 Models.py

```
import tensorflow.compat.v1 as tf
from layers import max_pooling, dropout, norm, conv2d, fc


def alexnet(inputs, num_classes, keep_prob):
    """Create alexnet model
    """
    x = tf.reshape(inputs, shape=[-1, 28, 28, 1])

    # first conv layer, downsampling layer, and normalization layer
    conv1 = conv2d(x, shape=(11, 11, 1, 96), padding='SAME',
name='conv1')
    pool1 = max_pooling(conv1, ksize=(2, 2), stride=(2, 2),
padding='SAME', name='pool1')
    norm1 = norm(pool1, radius=4, name='norm1')

    # second conv layer
    conv2 = conv2d(norm1, shape=(5, 5, 96, 256), padding='SAME',
name='conv2')
```

```
pool2 = max_pooling(conv2, ksize=(2, 2), stride=(2, 2),
padding='SAME', name='pool2')
    norm2 = norm(pool2, radius=4, name='norm2')

    # 3rd conv layer
    conv3 = conv2d(norm2, shape=(3, 3, 256, 384), padding='SAME',
name='conv3')
    # pool3 = max_pooling(conv3, ksize=(2, 2), stride=(2, 2),
padding='SAME', name='pool3')
    norm3 = norm(conv3, radius=4, name='norm3')

    # 4th conv layer
    conv4 = conv2d(norm3, shape=(3, 3, 384, 384), padding='SAME',
name='conv4')

    # 5th conv layer
    conv5 = conv2d(conv4, shape=(3, 3, 384, 256), padding='SAME',
name='conv5')
    pool5 = max_pooling(conv5, ksize=(2, 2), stride=(2, 2),
padding='SAME', name='pool5')
    norm5 = norm(pool5, radius=4, name='norm5')

    # first fully connected layer
    fc1 = tf.reshape(norm5, shape=(-1, 4*4*256))
    fc1 = fc(fc1, shape=(4*4*256, 4096), name='fc1')
    fc1 = dropout(fc1, keep_prob=keep_prob, name='dropout1')

    fc2 = fc(fc1, shape=(4096, 4096), name='fc2')
    fc2 = dropout(fc2, keep_prob=keep_prob, name='dropout2')

    # output logits value
    with tf.variable_scope('classifier') as scope:
        weights = tf.get_variable('weights', shape=[4096, num_classes],
initializer=tf.initializers.he_normal())
        biases = tf.get_variable('biases', shape=[num_classes],
```

```
initializer=tf.initializers.random_normal())
        # define output logits value
        logits = tf.add(tf.matmul(fc2, weights), biases,
name=scope.name + '_logits')

    return logits
```

7.3 Layers.py

```
import tensorflow.compat.v1 as tf
"""
from numpy.random import seed
seed(1)
from tensorflow import set_random_seed
set_random_seed(2)
"""
def max_pooling(inputs, ksize, stride, padding, name):
    """Create max pooling layer
    Args:
        inputs: float32 4D tensor
        ksize: a tuple of 2 int with (kernel_height, kernel_width)
        stride: a tuple
        padding: string. padding mode 'SAME', '
        name: string

    Returns:
        4D tensor of [batch_size, height, width, channels]
    """
```

```
with tf.variable_scope(name) as scope:
        value = tf.nn.max_pool(inputs, ksize=[1, ksize[0], ksize[1], 1],
strides=[1, stride[0], stride[1], 1], padding=padding,
name=scope.name)
    return value

def dropout(inputs, keep_prob, name):
    """Dropout layer

    Args:
        inputs: float32 4D tensor
        keep_prob: the probability of keep training sample
        name: layer name to define
    Returns:
        4D tensor of [batch_size, height, width, channels]
    """

    return tf.nn.dropout(inputs, rate=(1-keep_prob), name=name)


def norm(inputs, radius=4, name=None):
    """

    """
    with tf.variable_scope(name) as scope:
        value = tf.nn.lrn(inputs, depth_radius=radius, bias=1.0,
alpha=1e-4, beta=0.75, name=name)
    return value

# def batch_norm(inputs, name):
#     """batch normalization layer


#     """
```

```
def conv2d(inputs, shape, padding, name):
    """Create convolution 2D layer
    Args:
        inputs: float32. 4D tensor
        shape: the shape of kernel
        padding: string. padding mode 'SAME',
        name: corressponding layer's name
    Returns:
        Output 4D tensor
    """
    with tf.variable_scope(name) as scope:
        # get weights value and record a summary protocol buffer with
a histogram
        weights = tf.get_variable('weights', shape=shape,
initializer=tf.initializers.he_normal())
        tf.summary.histogram(scope.name + 'weights', weights)

        # get biases value and record a summary protocol buffer with a
histogram
        biases = tf.get_variable('biases', shape=shape[3],
initializer=tf.initializers.random_normal())
        tf.summary.histogram(scope.name + 'biases', biases)
        # compute convlotion W * X + b, activiation function relu
function
        outputs = tf.nn.conv2d(inputs, weights, strides=[1, 1, 1, 1],
padding=padding)
        outputs = tf.nn.bias_add(outputs, biases)
        outputs = tf.nn.relu(outputs, name=scope.name + 'relu')
    return outputs

def fc(inputs, shape, name):
    """Create fully collection layer
    Args:
        inputs: Float32. 2D tensor with shape [batch, input_units]
        shape: Int. a tuple with [num_inputs, num_outputs]
        name: sring. layer name
```

```
 Returns:
        Outputs fully collection tensor
    """

    with tf.variable_scope(name) as scope:
        weights = tf.get_variable('weights', shape = [shape[0], 
shape[1]], initializer=tf.initializers.he_normal())
        biases = tf.get_variable('biases', shape = [shape[1]], 
initializer=tf.initializers.random_normal())
        # outputs = tf.nn.xw_plus_b(inputs, weights, biases, name = 
scope.name)
        outputs = tf.add(tf.matmul(inputs, weights), biases, 
name=scope.name)

    return tf.nn.relu(outputs)
```

7.4 Evals.py

```
import tensorflow.compat.v1 as tf

def calc_loss_acc(labels, logits):
    """Function to compute loss value. Here, we used cross entropy
    Args:
        logits: 4D tensor. output tensor from segnet model, which is the 
output of softmax
        labels: true labels tensor
    Returns:
        loss (cross_entropy_mean), accuracy, predicts(logits with 
softmax)
    """
    # calc cross entropy mean  cross_entropy
    cross_entropy = 
tf.nn.softmax_cross_entropy_with_logits(labels=labels, logits=logits, 
name='cross_entropy')
```

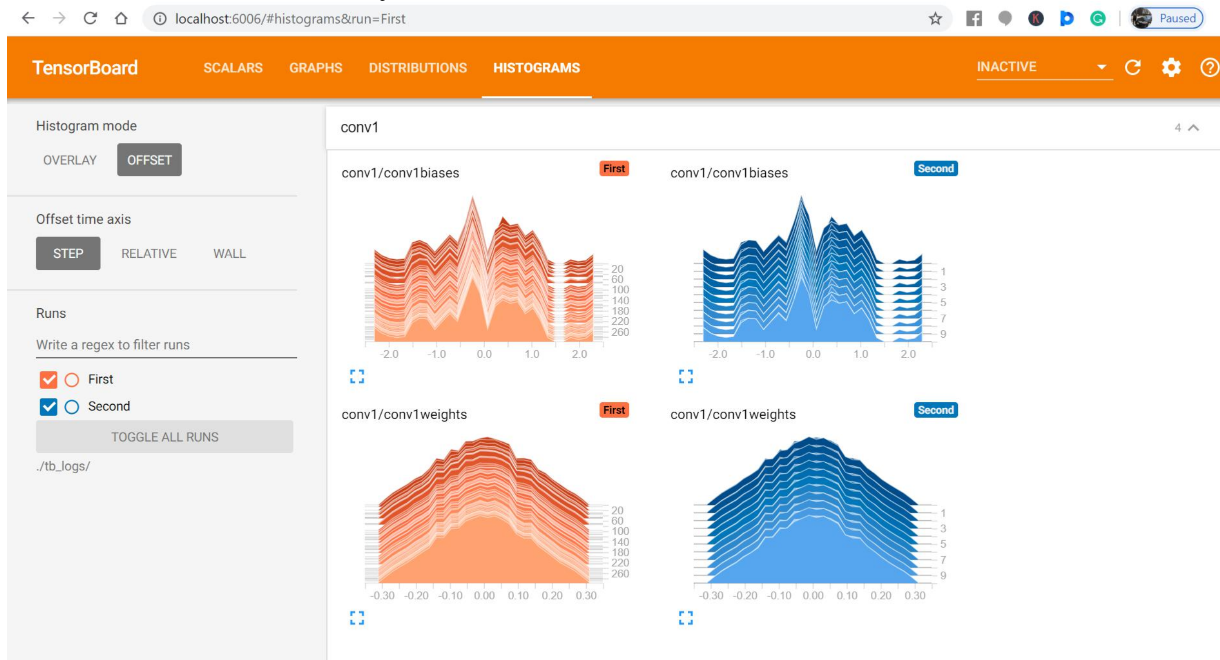
```
cross_entropy_mean = tf.reduce_mean(cross_entropy)
    tf.summary.scalar(name='loss', tensor=cross_entropy_mean)


    predicts = tf.equal(tf.argmax(logits, axis=1), tf.argmax(labels,
axis=1))

    accuracy = tf.reduce_mean(tf.cast(predicts, dtype=tf.float32))
    tf.summary.scalar(name='accuracy', tensor=accuracy)

    return cross_entropy_mean, accuracy, predicts


def train_op(total_loss, global_steps, base_learning_rate,
option='Adam'):
    """This function defines train optimizer
    Args:
        total_loss: the loss value
        global_steps: global steps is used to track how many batch had
been passed. In the training process, the initial value for global_steps
= 0, here
        global_steps=tf.Variable(0, trainable=False). then after one
batch of images passed, the loss is passed into the optimizer to
update the weight, then the global
        step increased by one.
        base_learning_rate: default value 0.1
    Returns:
        the train optimizer
    """

    # base_learning_rate = 0.01
    # get update operation
    update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
    with tf.control_dependencies(update_ops):
```

```
 if option == 'Adam':
          optimizer =
tf.train.AdamOptimizer(learning_rate=base_learning_rate)
          print("Running with Adam Optimizer with learning rate:",
base_learning_rate)
       elif option == 'SGD':
          # base_learning_rate = 0.01
          learning_rate_decay =
tf.train.exponential_decay(base_learning_rate, global_steps, 1000,
0.0005)
          optimizer =
tf.train.GradientDescentOptimizer(learning_rate_decay)
          print("Running with SGD Optimizer with learning rate:",
learning_rate_decay)
       else:
          raise ValueError('Optimizer is not recognized')

       grads = optimizer.compute_gradients(total_loss,
var_list=tf.trainable_variables())
       training_op = optimizer.apply_gradients(grads,
global_step=global_steps)

   return training_op
```

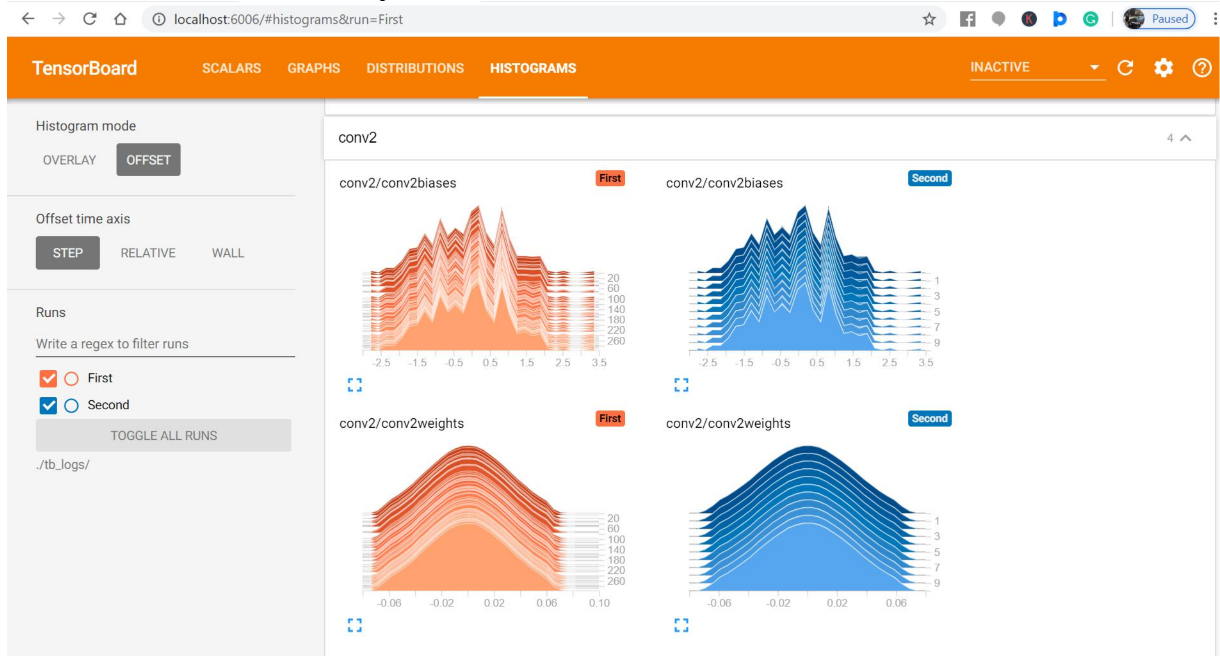
Chapter 8

Screen shots of Project

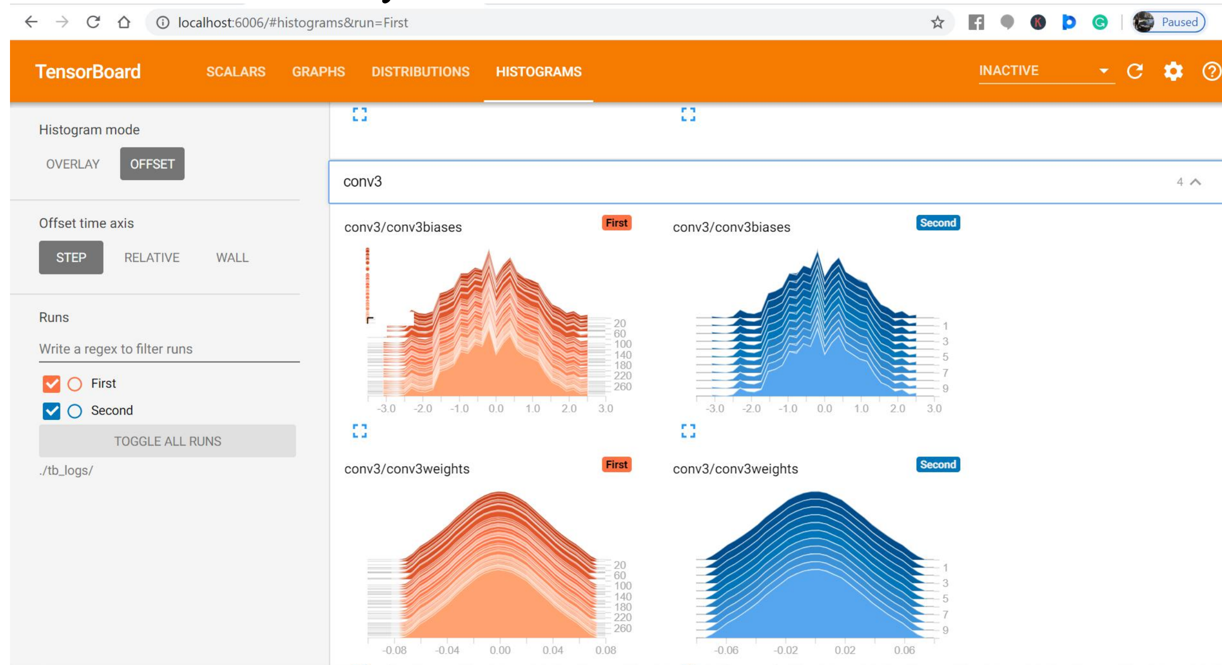
8.1 Convolution Layer 1



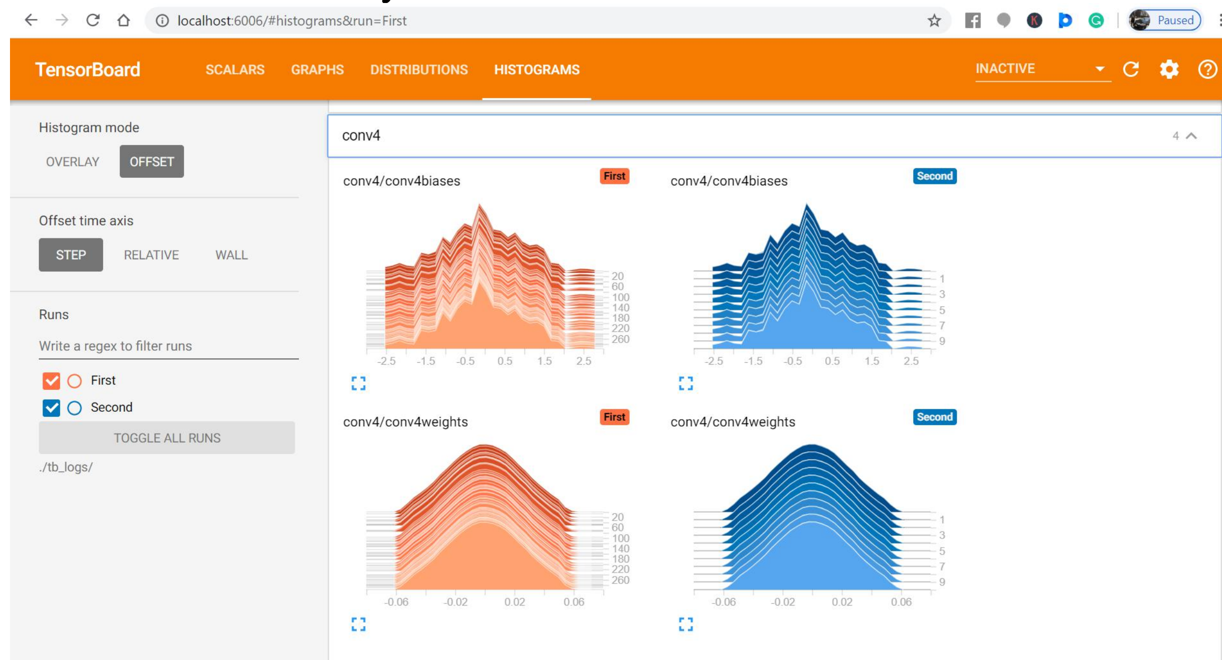
8.2 Convolution Layer 2



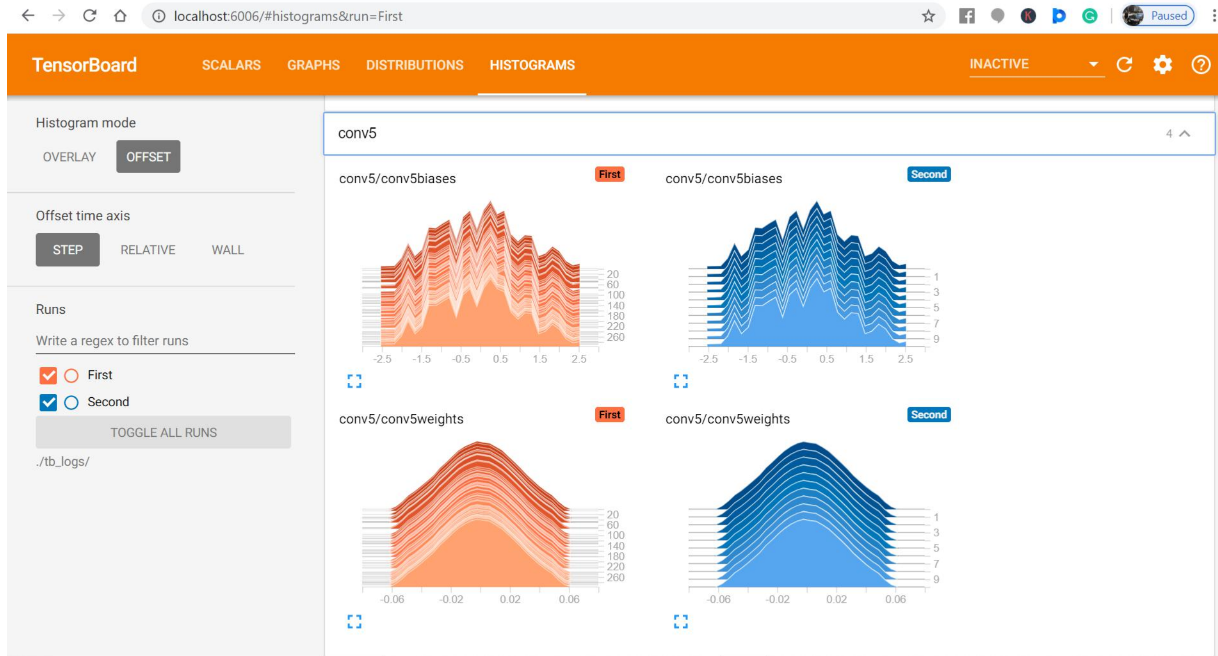
8.3 Convolution Layer 3



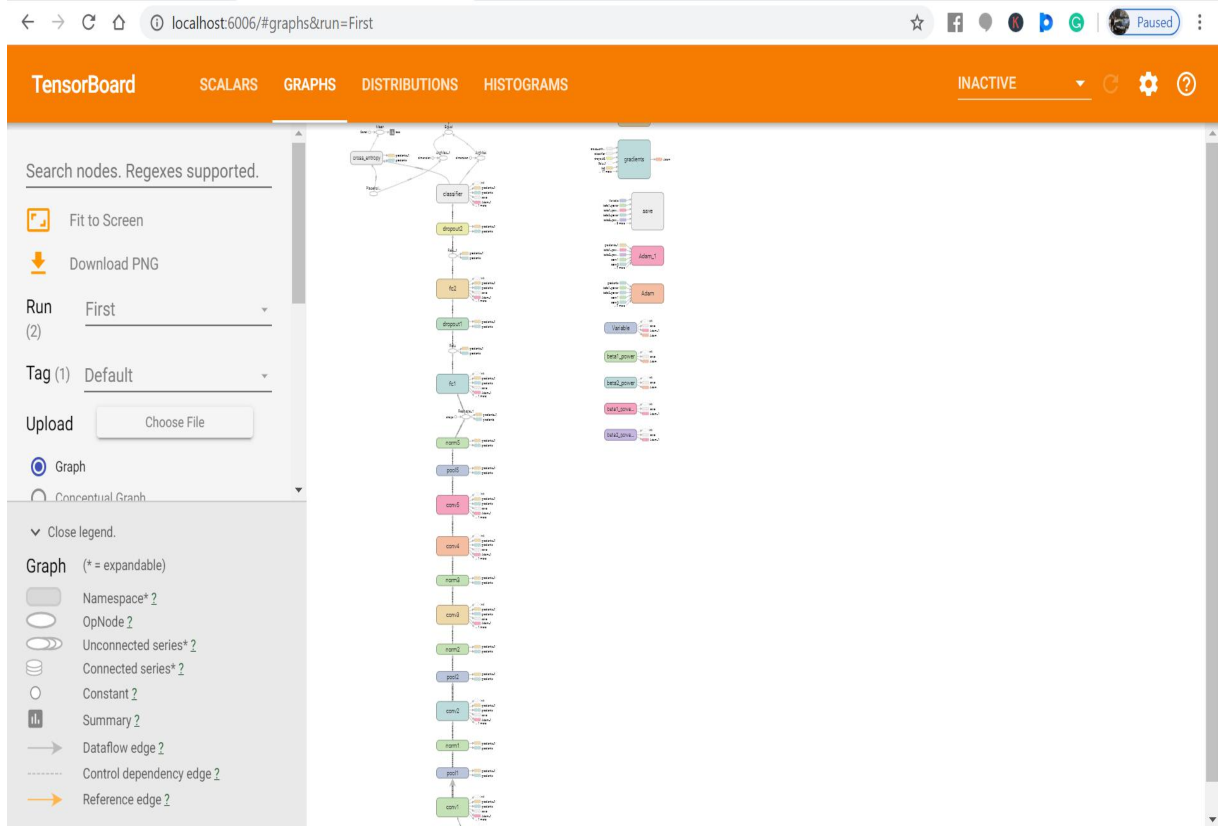
8.4 Convolution Layer 4



8.5 Convolution layer 5



8.6 The Different Layers



Chapter 10

Conclusion and Future Scope

10.1 Conclusion

The project implementation will be useful in various domains of Astronomy as position detection with accuracy is used in many fields. Thus Deep Learning was a useful tool in the error detection and as it is an upcoming field in the Computer Science Field there would be a lot of scope of improvement in this area.

10.2 Future Scope

There are chances of further improving the accuracy of the model as better models come in future and thus would be more trustworthy in the future.It can be used by space organizations in navigation systems and locating items in the space.

References

[1] http://astronomy.swin.edu.au/cosmos/E/Equatorial+Coordinate+System
[2] https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19700008007.pdf
[3] https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2
[4] https://www.learnopencv.com/understanding-alexnet/
[5] http://cdsweb.u-strasbg.fr/about
[6] https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/
[7] https://stats.stackexchange.com/questions/369227/rgb-images-as-input-to-cnn
[8] https://www.geeksforgeeks.org/python-all-permutations-of-a-string-in-lexicographical-order-without-using-recursion/
[9] https://github.com/zhaoqi19/MNIST-AlexNet-Using-Tensorflow/tree/master/image

SAMPLE INDIVIDUAL CONTRIBUTION REPORT:

MACHINE LEARNING APPLICATION IN STELLAR FIELD IDENTIFICATION

HARSHITA ARYA
1605509

Abstract: The project is based on the process of error detection in the positioning of the stellar fields and is used for its application in Astonomy.The Deep Learning tool used was AlexNet and the implementation was carried out in Python using tensorflow backed.

Individual contribution and findings: Since I was the only student I implemented the project under the guidence of my project mentor and my guide in the institute that I was carrying out the project.I used the books available and the resources available online for my findings

Individual contribution to project report preparation: Since I was the only one doing the project I did the report preparation

Individual contribution for project presentation and demonstration: As mentioned I was the only one doing this project I did the project demonstration.

Full Signature of Supervisor:

Full signature of the student:
Harshita Arya