**Chapter 1**

**Introduction**

Reading text in Natural Images is significant in a variety of advanced computer vision applications, such as image and video retrieval, scene understanding and visual assistance, since text in images usually conveys valuable information. Hence, detection and recognizing text in scene images has received increasing attention in this community. Though extensively studied in recent years, text detection in unconstrained environments is still quite challenging due to a number of factors, such as high variation in character font, size, color, orientation as well as complicated background and non-uniform illumination.

Previous works for scene text detection based on sliding windows and connected component analysis have become mainstream in this domain. Sliding windows based methods localize text regions by shifting a multi-scaled classification window. This exhaustive search is computationally inefficient though it achieves high recall rates. Methods based on connected components extract individual characters through connected component analysis followed by grouping and refinement strategy. Additionally, false alarm removing may be performed to remove non-text components. Stroke Width Transform (SWT) and Maximally Stable Extremal Region (MSER) are two representative techniques, particularly methods based on MSER achieved the state-of-the-art performance on CHAR74K dataset. However, the MSER algorithms extract massive repeating non-text components which will be constrained by false-removing and refinement rules. These methods are also incapable of detecting characters distorted by noise or background.

More recently, several deep learning based approaches have been developed for scene text detection owing to deep model feature representations. These models building on convolutional neural networks (CNN) compute high-level deep features from image patches or proposals for text/non-text classification. These methods are also restricted by region proposal methods and the discriminative power of CNN classifiers.

In this project, we propose a robust approach which combines the advantages of both MSER and CNN feature representations. Our contributions can be summarized into three points. First, a saliency enhanced-MSER, which is an extension of the well-known MSER algorithm by incorporating saliency detection methods, is proposed as character candidate extractor on three channels of the image to ensure a high recall rate. The second contribution is a novel text filtering pipeline with a deep CNN. In the classification stage, we train a powerful convolutional neural network which incorporates pixel-level and character-level information.

**1.1 Applications**

**1.1.1 Practical Applications:**

In recent years, OCR (Optical Character Recognition) technology has been applied throughout the entire spectrum of industries, revolutionizing the document management process. OCR has enabled scanned documents to become more than just image files, turning into fully searchable documents with text content that is recognized by computers. With the help of OCR, people no longer need to manually retype important documents when entering them into electronic databases. Instead, OCR extracts relevant information and enters it automatically. The result is accurate, efficient information processing in less time.

**1.1.2 Banking:**

The uses of OCR vary across different fields. One widely known application is in banking, where OCR is used to process checks without human involvement. A check can be inserted into a machine, the writing on it is scanned instantly, and the correct amount of money is transferred. This technology has nearly been perfected for printed checks, and is fairly accurate for handwritten checks as well, though it occasionally requires manual confirmation. Overall, this reduces wait times in many banks.

**1.1.3 Legal:**

In the legal industry, there has also been a significant movement to digitize paper documents. In order to save space and eliminate the need to sift through boxes of paper files, documents are being scanned and entered into computer databases. OCR further simplifies the process by making documents text-searchable, so that they are easier to locate and work with once in the database. Legal professionals now have fast, easy access to a huge library of documents in electronic format, which they can find simply by typing in a few keywords.

**1.1.4 Healthcare:**

Healthcare has also seen an increase in the use of OCR technology to process paperwork. Healthcare professionals always have to deal with large volumes of forms for each patient, including insurance forms as well as general health forms. To keep up with all of this information, it is useful to input relevant data into an electronic database that can be accessed as necessary. Form processing tools, powered by OCR, are able to extract information from forms and put it into databases, so that every patient's data is promptly recorded. As a result, healthcare providers can focus on delivering the best possible service to every patient.

**1.1.5 OCR in Other Industries:**

OCR is widely used in many other fields, including education, finance, and government agencies. OCR has made countless texts available online, saving money for students and allowing knowledge to be shared. Invoice imaging applications are used in many businesses to keep track of financial records and prevent a backlog of payments from piling up. In government agencies and independent organizations, OCR simplifies data collection and analysis, among other processes. As the technology continues to develop, more and more applications are found for OCR technology, including increased use of handwriting recognition. Furthermore, other technologies related to OCR, such as barcode recognition, are used daily in retail and other industries. To learn more about OCR solutions for your office, you can download a free trial of Maestro Recognition Server, CVISION's OCR toolkit, or Trapeze, our automated form-processing solution.

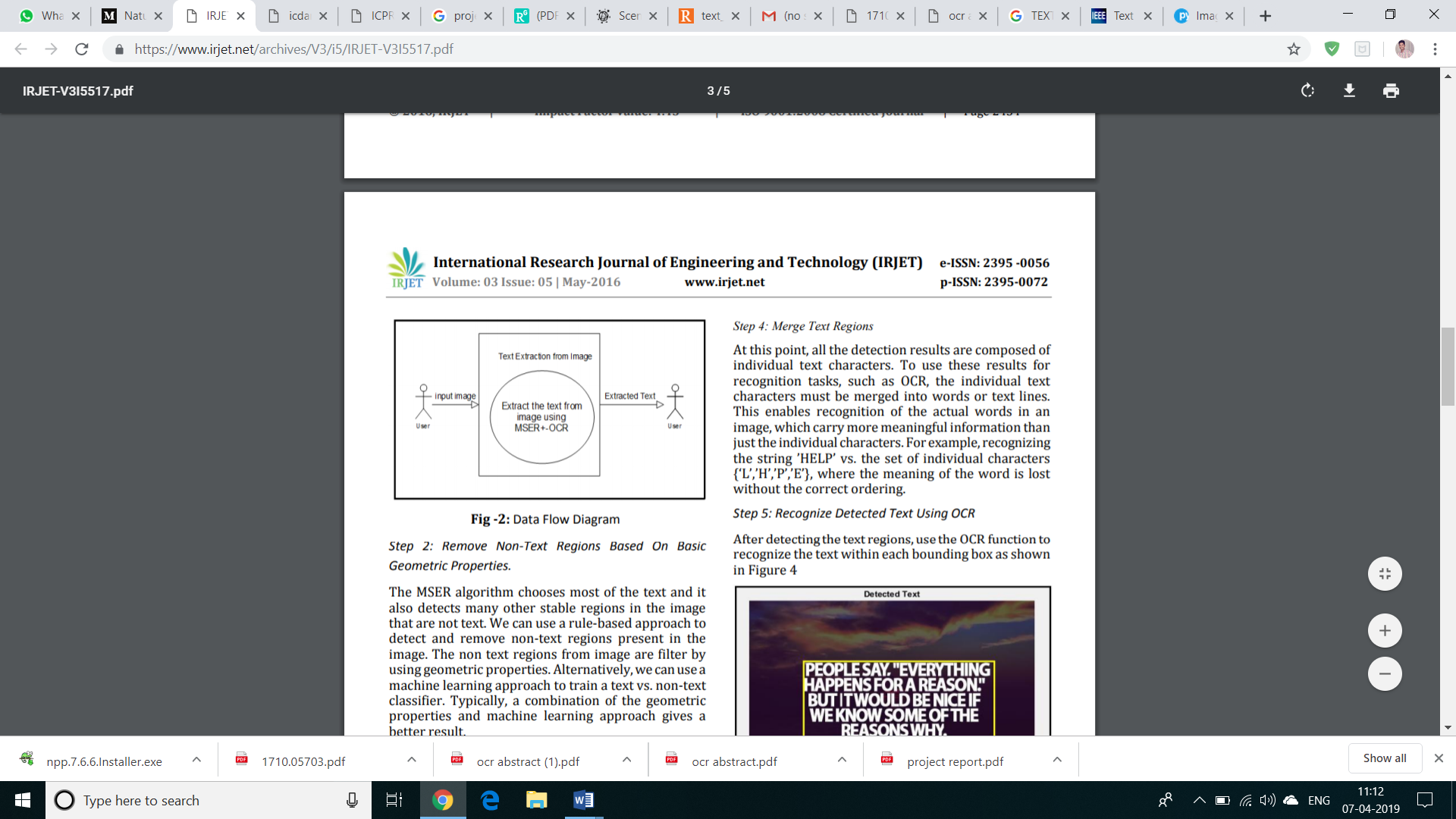
**Chapter 2**

**Literature Survey**

Character recognition is not a new problem but its roots can be traced back to systems before the inventions of computers. The earliest OCR systems were not computers but mechanical devices that were able to recognize characters, but very slow speed and low accuracy. In 1951, M. Sheppard invented a reading and robot GISMO that can be considered as the earliest work on modern OCR [1]. GISMO can read musical notations as well as words on a printed page one by one. However, it can only recognize 23 characters. The machine also has the capability to could copy a typewritten page. J. Rainbow, in 1954, devised a machine that can read uppercase typewritten English characters, one per minute. The early OCR systems were criticized due to errors and slow recognition speed. Hence, not much research efforts were put on the topic during 60’s and 70’s. The only developments were done on government agencies and large corporations like banks, newspapers and airlines etc. Because of the complexities associated with recognition, it was felt that three should be standardized OCR fonts for easing the task of recognition for OCR. Hence, OCRA and OCRB were developed by ANSI and EMCA in 1970, that provided comparatively acceptable recognition rates[2] . During the past thirty years, substantial research has been done on OCR. This has lead to the emergence of document image analysis (DIA), multi-lingual, handwritten and omni-font OCRs [2]. Despite these extensive research efforts, the machine’s ability to reliably read text is still far below the human. Hence, current OCR research is being done on improving accuracy and speed of OCR for diverse style documents printed/ written in unconstrained environments. There has not been availability of any open source or commercial software available for complex languages like Urdu or Sindhi etc.

**Chapter 3**

**Methodology and Work Description**



Data Flow Diagram of entire Project

**3.1 THE TEXT DETECTION ALGORITHM**

In this section, we describe the text detection algorithm that is MSER (Maximally Stable Extremal Region) algorithm. MSER is a method for text detection, blob detection in images. The MSER algorithm extracts number of co-variant regions from image. MSER is based on the idea of taking regions which stay nearly the same through a wide range of thresholds. All the pixels above or equal to a given threshold are black and all the pixels below a given threshold are white. MSER uses two important properties to remove non text regions from image first is Geometric Properties and another is Stroke Width Variation Properties. To use MSER algorithm we first summarized the common attributes of text as

a) Text in image always contains lots of edges.

b) The width of text is larger than height.

c) Text is bounded in size.

d) Text has special texture but this texture is irregular.

**The below Diagram describes How the MSER works:**



Before Feeding to MSER

­

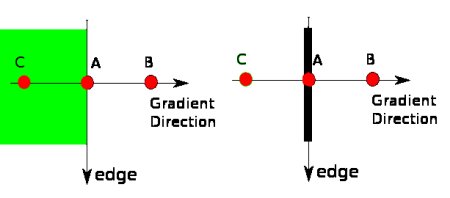
After Feeding to MSER

**3.2 Non-maximum Suppression:**

Non-maximum suppression (NMS) has been widely used in several key aspects of computer vision and is an integral part of many proposed approaches in detection, might it be edge, corner or object detection. Its necessity stems from the imperfect ability of detection algorithms to localize the concept of interest, resulting in groups of several detections near the real location. In the context of object detection, approaches based on sliding windows typically produce multiple windows with high scores close to the correct location of objects. This is a consequence of the generalization ability of object detectors, the smoothness of the response function and visual correlation of close-by windows. This relatively dense output is generally not satisfying for understanding the content of an image. As a matter of fact, the number of window hypotheses at this step is simply uncorrelated with the real number of objects in the image. The goal of NMS is therefore to retain only one window per group, corresponding to the precise local maximum of the response function, ideally obtaining only one detection per object. Consequently, NMS also has a large positive impact on performance measures that penalize double detections .

The most common approach for NMS consists of a greedy iterative procedure , which we refer to as Greedy NMS. The procedure starts by selecting the best scoring window and assuming that it indeed covers an object. Then, the windows that are too close to the selected window are suppressed. Out of the remaining windows, the next top-scoring one is selected, and the procedure is repeated until no more windows remain. This procedure involves defining a measure of similarity between windows and setting a threshold for suppression. These definitions vary substantially from one work to another, but typically they are manually designed. Greedy NMS, although relatively fast, has a number of downsides, as illustrated in Fig. 1. First, by suppressing everything within the neighborhood with a lower confidence, if two or more objects are close to each other, all but one of them will be suppressed. Second, Greedy NMS always keeps the detection with the highest confidence even though in some cases another detection in the surrounding might provide a better fit for the true object. Third, it returns all the bounding-boxes which are not suppressed, even though many could be ignored due to a relatively low confidence or the fact that they are sparse in a subregion within the image

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. Check the image below:



Direction of gradient

Point A is on the edge ( in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed ( put to zero).

In short, the result you get is a binary image with “thin edges”.

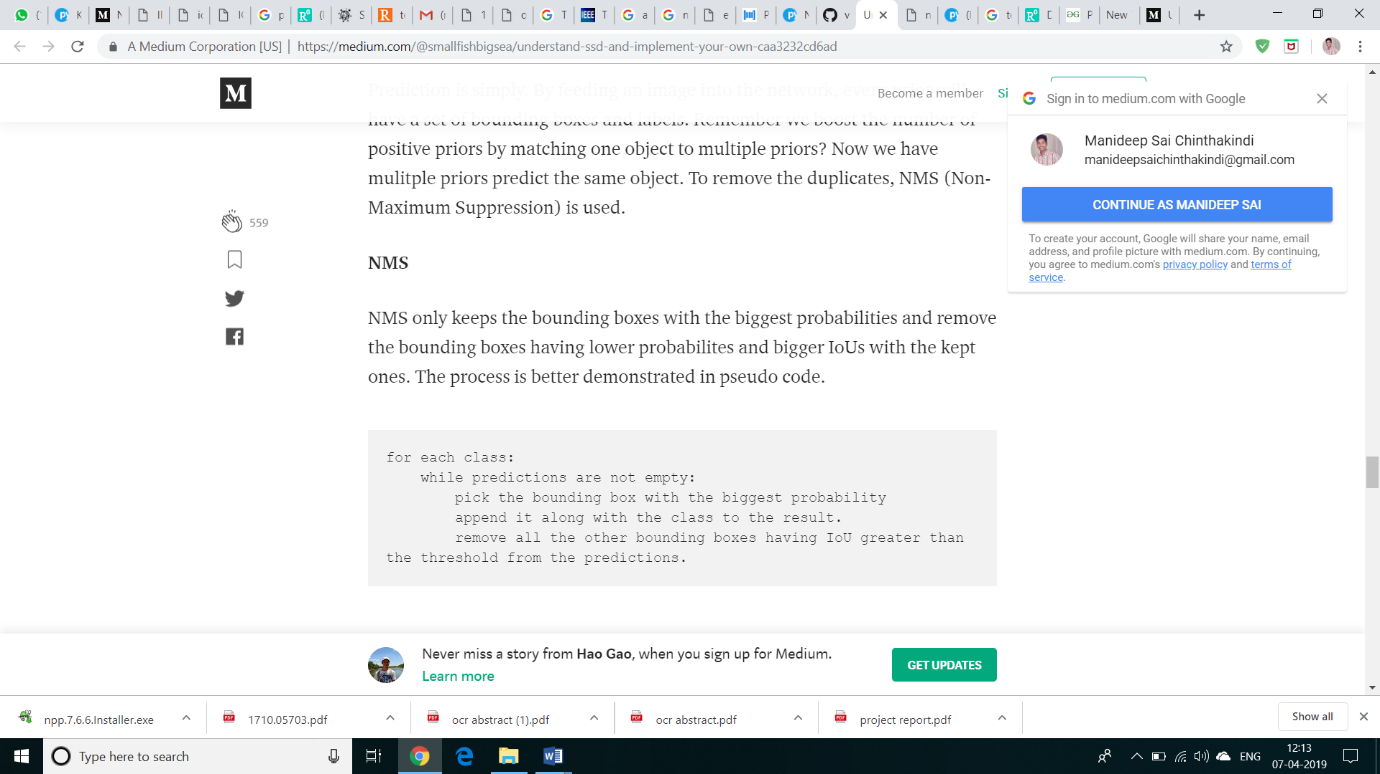


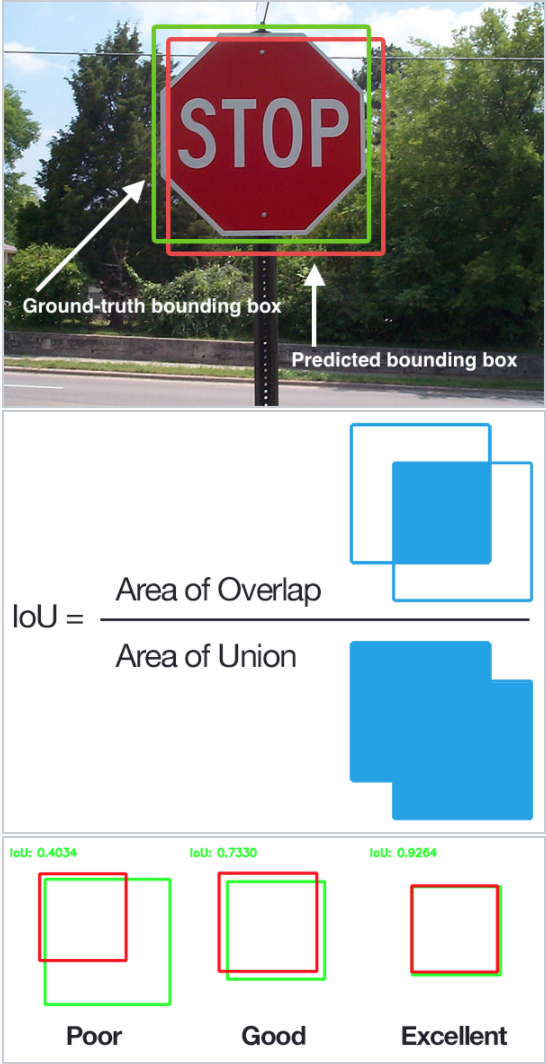
Before Applying NMS



After Applying NMS

**Pseudo Code For NMS**:



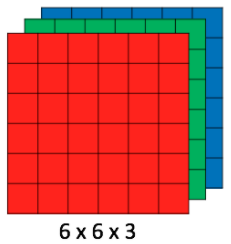


IoU-Intersection over Union

**3.3 Convolutional Neural Network(CNN):**

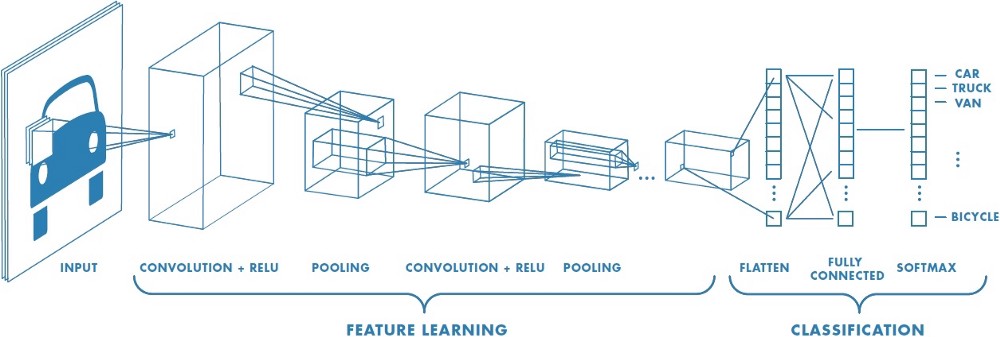
In neural networks, Convolutional neural network (ConvNets or CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

CNN image classifications takes an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see h x w x d( h = Height, w = Width, d = Dimension ). Eg., An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of grayscale image.



Matrix of greyscale image

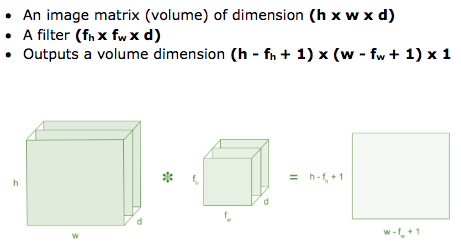
Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernals), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.



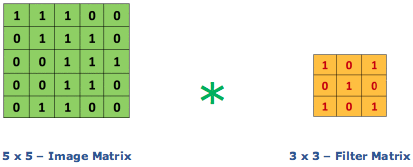
Flow of CNN Process

**Convolution Layer:**

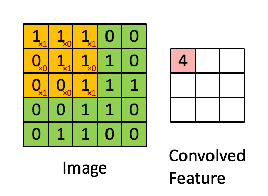
Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel



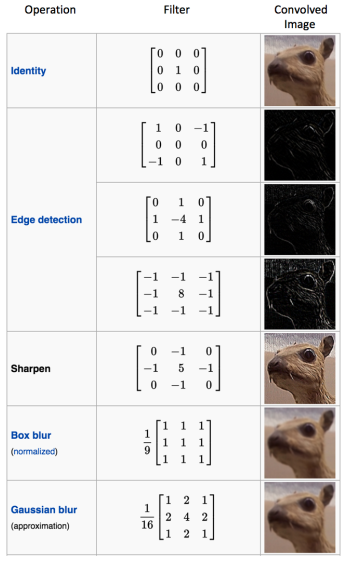
Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below



Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called **“Feature Map”** as output shown in below



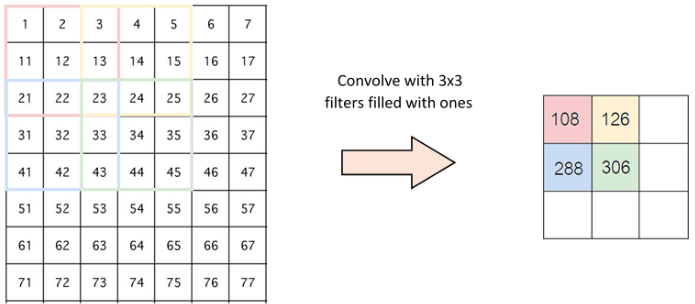
Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters (Kernels).



Types of filters

**Strides:**

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.



**Padding:**

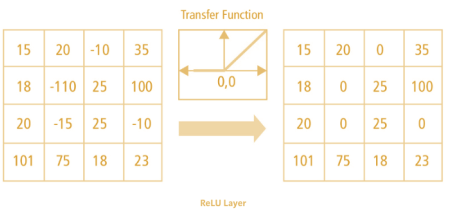
Sometimes filter does not fit perfectly fit the input image. We have two options:

* Pad the picture with zeros (zero-padding) so that it fits
* Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

**Non Linearity (ReLU):**

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is ***ƒ(x) = max(0,x).***

Why ReLU is important : ReLU’s purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.



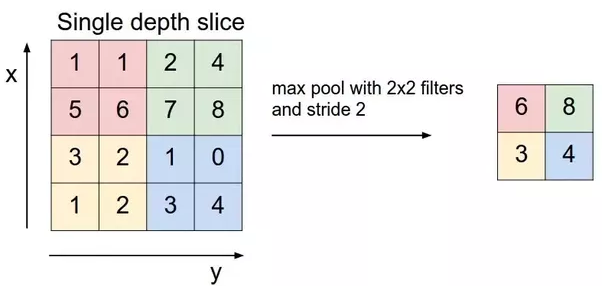
There are other non linear functions such as tanh or sigmoid can also be used instead of ReLU. Most of the data scientists uses ReLU since performance wise ReLU is better than other two.

**Pooling Layer:**

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains the important information. Spatial pooling can be of different types:

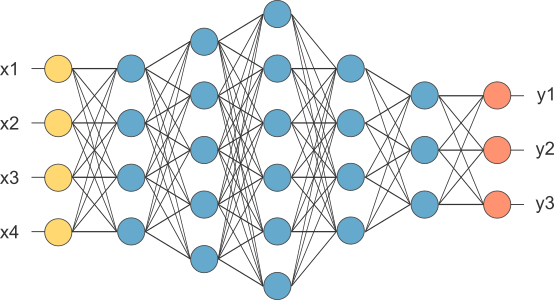
* Max Pooling
* Average Pooling
* Sum Pooling

Max pooling take the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

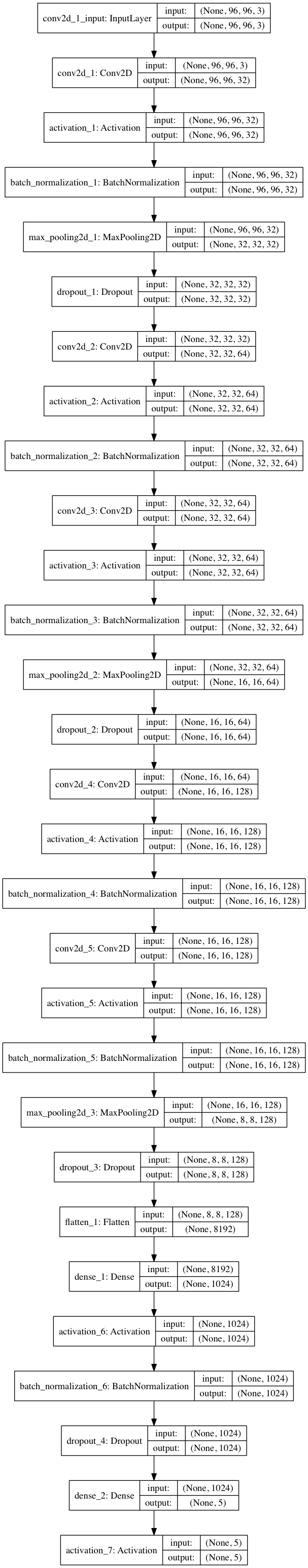


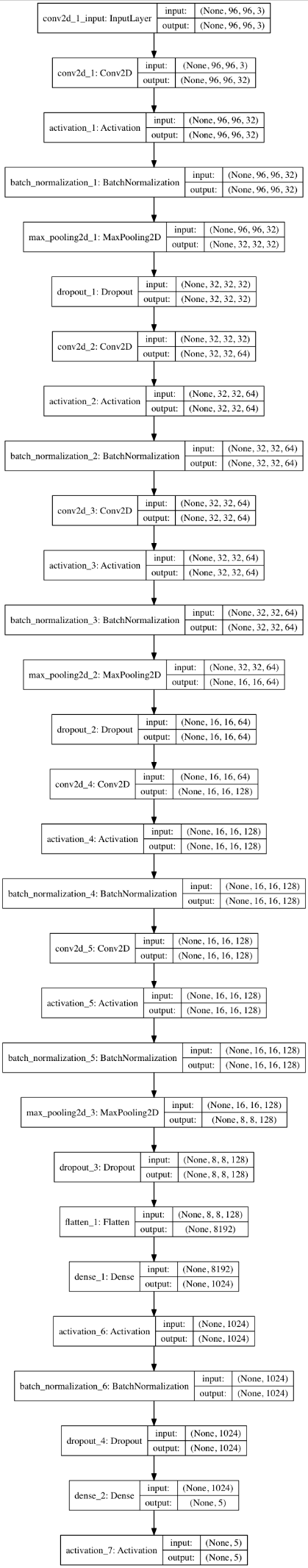
**Fully Connected Layer:**

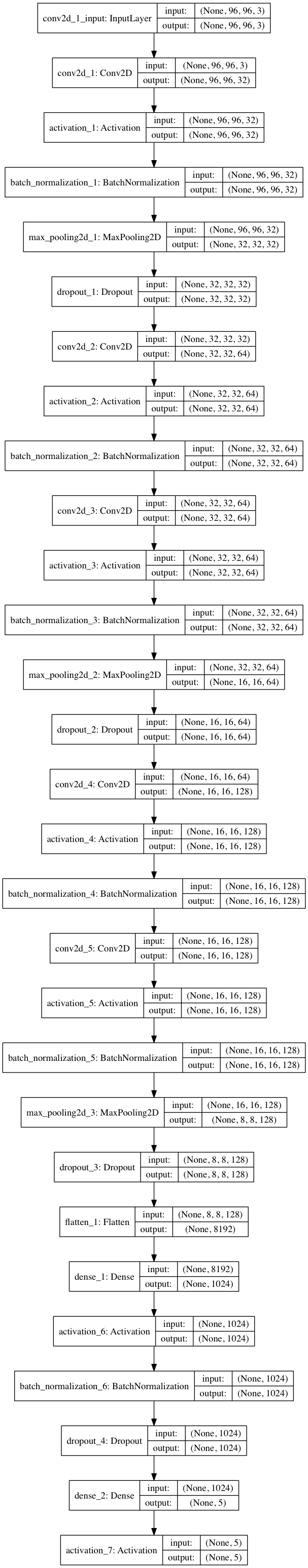
The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like neural network.



In the above diagram, feature map matrix will be converted as vector (x1, x2, x3, …). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the outputs as cat, dog, car, truck etc.,

**The CNN Architechture We are going to use is similar to vggnet:**





**Chapter 4**

**Tools and Technology Used:**

**4.1 SOFTWARE REQUIREMENT:**

Operating System: Windows 95/98/XP or higher.

Anaconda.(Miniconda)

Python IDE : Jupyter Notebook

Keras

Tensorflow

**4.2 HARDWARE REQUIREMENT**:

Screen resolution: 800X600 or Higher.

Support for printer that is, appropriate devices are installed and connected printer will be required for printing of the reports.

Desktop system, Handheld system-not a concern, as it will be possible to run the application on any of these

Minimum 500 GB of Hard disk space .Graphic Card AMD/NVIDIA(Recommended)

**Chapter 5**

**Implemantation and Coding**

This Section contains the codes of this project.We implemented problem statement using python 3.6. The editor used was Anaconda (Miniconda).For visualization and graphs we have used standard matplotlib Library of python.For implementing Neural Networks we have used Keras Library with Tensorflow as the backend.

**A. Training the Neural Network**

from keras.models import Sequential

from keras.layers.normalization import BatchNormalization

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

from keras.layers.core import Activation

from keras.layers.core import Flatten

from keras.layers.core import Dropout

from keras.layers.core import Dense

from keras import backend as K

class SmallerVGGNet:

@staticmethod

def build(width, height, depth, classes):

model = Sequential()

inputShape = (height, width, depth)

chanDim = -1

if K.image\_data\_format() == "channels\_first":

inputShape = (depth, height, width)

chanDim = 1

model.add(Conv2D(32, (3, 3), padding="same",

input\_shape=inputShape))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(MaxPooling2D(pool\_size=(3, 3)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(64, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

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

model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(128, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

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

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(1024))

model.add(Activation("relu"))

model.add(BatchNormalization())

model.add(Dropout(0.5))

model.add(Dense(classes))

model.add(Activation("softmax"))

return model

**A.1**.Import the required libraries and defines the required classes

import matplotlib

matplotlib.use("Agg")

from keras.preprocessing.image import ImageDataGenerator

from keras.optimizers import Adam

from keras.preprocessing.image import img\_to\_array

from sklearn.preprocessing import LabelBinarizer

from sklearn.model\_selection import train\_test\_split

from classes.smallervggnet import SmallerVGGNet

import matplotlib.pyplot as plt

from imutils import paths

import numpy as np

import argparse

import random

import pickle

import cv2

import os

ap = argparse.ArgumentParser()

ap.add\_argument("-d", "--dataset", required=True,

help="path to input dataset (i.e., directory of images)")

ap.add\_argument("-m", "--model", required=True,

help="path to output model")

ap.add\_argument("-l", "--labelbin", required=True,

help="path to output label binarizer")

ap.add\_argument("-p", "--plot", type=str, default="plot.png",

help="path to output accuracy/loss plot")

args = vars(ap.parse\_args())

EPOCHS = 100

INIT\_LR = 1e-3

BS = 32

IMAGE\_DIMS = (96, 96, 3)

data = []

labels = []

print("[INFO] loading images...")

imagePaths = sorted(list(paths.list\_images(args["dataset"])))

random.seed(42)

random.shuffle(imagePaths)

for imagePath in imagePaths:

image = cv2.imread(imagePath)

image = cv2.resize(image, (IMAGE\_DIMS[1], IMAGE\_DIMS[0]))

image = img\_to\_array(image)

data.append(image)

label = imagePath.split(os.path.sep)[-2]

labels.append(label)

data = np.array(data, dtype="float") / 255.0

labels = np.array(labels)

print("[INFO] data matrix: {:.2f}MB".format(

data.nbytes / (1024 \* 1000.0)))

lb = LabelBinarizer()

labels = lb.fit\_transform(labels)

(trainX, testX, trainY, testY) = train\_test\_split(data,

labels, test\_size=0.2, random\_state=42)

aug = ImageDataGenerator(rotation\_range=25, width\_shift\_range=0.1,

height\_shift\_range=0.1, shear\_range=0.2, zoom\_range=0.2,

horizontal\_flip=True, fill\_mode="nearest")

print("[INFO] compiling model...")

model = SmallerVGGNet.build(width=IMAGE\_DIMS[1], height=IMAGE\_DIMS[0],

depth=IMAGE\_DIMS[2], classes=len(lb.classes\_))

opt = Adam(lr=INIT\_LR, decay=INIT\_LR / EPOCHS)

model.compile(loss="categorical\_crossentropy", optimizer=opt,

metrics=["accuracy"])

print("[INFO] training network...")

H = model.fit\_generator(

aug.flow(trainX, trainY, batch\_size=BS),

validation\_data=(testX, testY),

steps\_per\_epoch=len(trainX) // BS,

epochs=EPOCHS, verbose=1)

print("[INFO] serializing network...")

model.save(args["model"])

print("[INFO] serializing label binarizer...")

f = open(args["labelbin"], "wb")

f.write(pickle.dumps(lb))

f.close()

**A.2**.Trains the model on the dataset

**B. Recognising Text from Images**

import numpy as np

def non\_max\_suppression\_slow(boxes, overlapThresh):

if len(boxes) == 0:

return []

pick = []

x1 = boxes[:,0]

y1 = boxes[:,1]

x2 = boxes[:,2]

y2 = boxes[:,3]

area = (x2 - x1 + 1) \* (y2 - y1 + 1)

idxs = np.argsort(y2)

while len(idxs) > 0:

last = len(idxs) - 1

i = idxs[last]

pick.append(i)

suppress = [last]

for pos in range(0, last):

j = idxs[pos]

xx1 = max(x1[i], x1[j])

yy1 = max(y1[i], y1[j])

xx2 = min(x2[i], x2[j])

yy2 = min(y2[i], y2[j])

w = max(0, xx2 - xx1 + 1)

h = max(0, yy2 - yy1 + 1)

overlap = float(w \* h) / area[j]

if overlap > overlapThresh:

suppress.append(pos)

idxs = np.delete(idxs, suppress)

return boxes[pick]

**B.1**.Import the required libraries and defines the required classes

**C. Final Outcome**

from keras.preprocessing.image import img\_to\_array

from keras.models import load\_model

import numpy as np

import argparse

import imutils

import pickle

import cv2

import os

from classes.nms import non\_max\_suppression\_slow

ap = argparse.ArgumentParser()

ap.add\_argument("-m", "--model", required=True,

help="path to trained model model")

ap.add\_argument("-l", "--labelbin", required=True,

help="path to label binarizer")

ap.add\_argument("-i", "--image", required=True,

help="path to input image")

args = vars(ap.parse\_args())

img = cv2.imread(args["image"])

mser = cv2.MSER\_create()

img = cv2.resize(img, (img.shape[1]\*2, img.shape[0]\*2))

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

vis = img.copy()

rects = []

regions = mser.detectRegions(gray)

hulls = [cv2.convexHull(p.reshape(-1, 1, 2)) for p in regions[0]]

for i, contour in enumerate(hulls):

x,y,w,h = cv2.boundingRect(contour);rects.append((x, y, x+w, y+h))

pick = non\_max\_suppression\_slow(np.array(rects), 0.3)

text=""

lb = pickle.loads(open(args["labelbin"], "rb").read())

model = load\_model(args["model"])

i=0

results = []

for (startX, startY, endX, endY) in pick:

cv2.rectangle(vis, (startX, startY), (endX, endY), (0, 255, 0), 2)

image = img[startY: endY,startX: endX].copy()

#cv2.imwrite('{}.png'.format(i), img[startY:endY,startX:endX])

#cv2.imshow("cropped", image)

#cv2.waitKey(0)

#cv2.imshow("fsk",image)

#cv2.waitKey(0)

image = cv2.resize(image, (96, 96))

image = image.astype("float") / 255.0

image = img\_to\_array(image)

image = np.expand\_dims(image, axis=0)

proba = model.predict(image)[0]

idx = np.argmax(proba)

label = lb.classes\_[idx]

filename = args["image"][args["image"].rfind(os.path.sep) + 1:]

#print(label)

if (label=="Sample001"):

label="0"

if (label=="Sample002"):

label="1"

if (label=="Sample003"):

label="2"

if (label=="Sample004"):

label="3"

if (label=="Sample005"):

label="4"

if (label=="Sample006"):

label="5"

if (label=="Sample007"):

label="6"

if (label=="Sample008"):

label="7"

if (label=="Sample009"):

label="8"

if (label=="Sample010"):

label="9"

if (label=="Sample011"):

label="A"

if (label=="Sample012"):

label="B"

if (label=="Sample013"):

label="C"

if (label=="Sample014"):

label="D"

if (label=="Sample015"):

label="E"

if (label=="Sample016"):

label="F"

if (label=="Sample017"):

label="G"

if (label=="Sample018"):

label="H"

if (label=="Sample019"):

label="I"

if (label=="Sample020"):

label="J"

if (label=="Sample021"):

label="K"

if (label=="Sample022"):

label="L"

if (label=="Sample023"):

label="M"

if (label=="Sample024"):

label="N"

if (label=="Sample025"):

label="O"

if (label=="Sample026"):

label="P"

if (label=="Sample027"):

label="Q"

if (label=="Sample028"):

label="R"

if (label=="Sample029"):

label="S"

if (label=="Sample030"):

label="T"

if (label=="Sample031"):

label="U"

if (label=="Sample032"):

label="V"

if (label=="Sample033"):

label="W"

if (label=="Sample034"):

label="X"

if (label=="Sample035"):

label="Y"

if (label=="Sample036"):

label="Z"

if (label=="Sample037"):

label="a"

if (label=="Sample038"):

label="b"

if (label=="Sample039"):

label="c"

if (label=="Sample040"):

label="d"

if (label=="Sample041"):

label="e"

if (label=="Sample042"):

label="f"

if (label=="Sample043"):

label="g"

if (label=="Sample044"):

label="h"

if (label=="Sample045"):

label="i"

if (label=="Sample046"):

label="j"

if (label=="Sample047"):

label="k"

if (label=="Sample048"):

label="l"

if (label=="Sample049"):

label="m"

if (label=="Sample050"):

label="n"

if (label=="Sample051"):

label="o"

if (label=="Sample052"):

label="p"

if (label=="Sample053"):

label="q"

if (label=="Sample054"):

label="r"

if (label=="Sample055"):

label="s"

if (label=="Sample056"):

label="t"

if (label=="Sample057"):

label="u"

if (label=="Sample058"):

label="v"

if (label=="Sample059"):

label="w"

if (label=="Sample060"):

label="x"

if (label=="Sample061"):

label="y"

if (label=="Sample062"):

label="z"

print (label)

cv2.putText(vis, label, (startX, startY - 20),cv2.FONT\_HERSHEY\_SIMPLEX, 1.2, (0, 0, 255), 3)

results.append(((startX, startY, endX, endY), label))

i=i+1

text = text + label

results = sorted(results, key=lambda r:r[0][1])

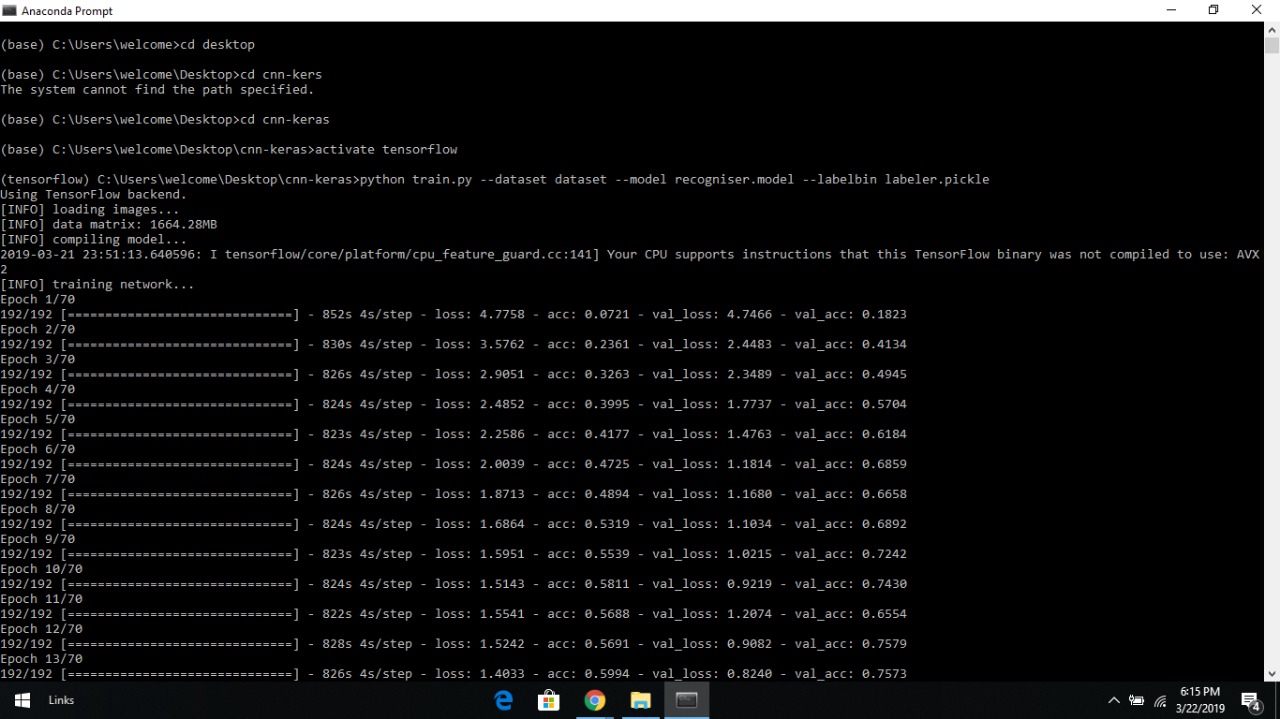
vis = imutils.resize(vis, width=400)

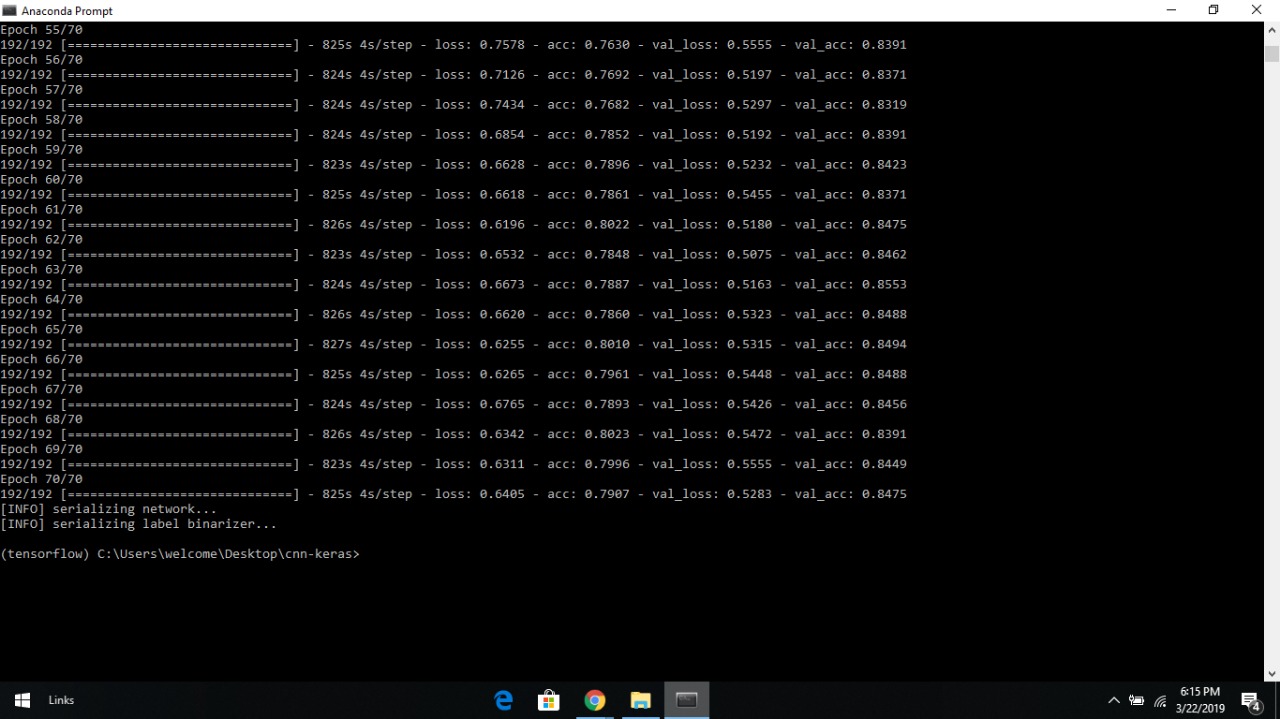
cv2.imshow("Output", vis)

cv2.waitKey(0)

**Chapter 6**

**Result Analysis**





Looking at the output of our training script we see that our Keras CNN obtained:

**84.91%** classification accuracy on the *training set*

And **85.75%** accuracy on the *testing set*

**The training loss/accuracy plot follows**:



As you can see in **Figure** , WE trained the model for 70 epochs and achieved low loss with limited overfitting. With additional training data we could obtain higher accuracy as well.

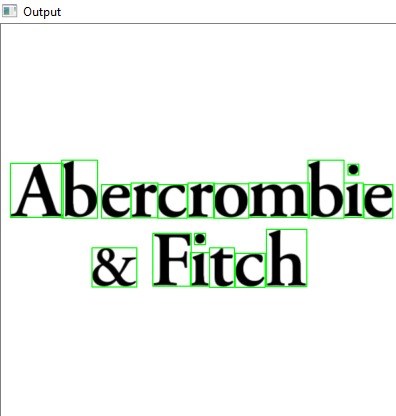
**The Results after running the Image through the CNN:**



Looking at the output of our training script we see that our Keras CNN obtained:

That the given image is 6 with 99.82% probability.

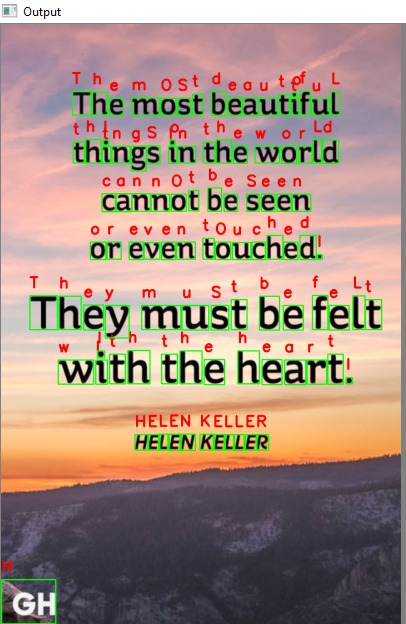
**The Results after running the Image through the MSER function:**



Looking at the output of our script we see that our code obtained:

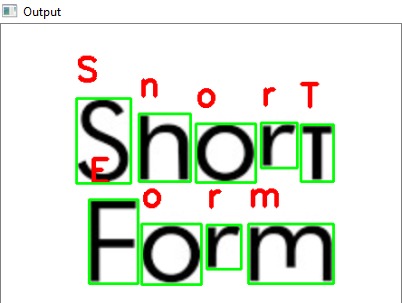
Image with all the bounding boxes where text is present.

**The Results after running the Image through the model:**



Looking at the output of our script we see that our Keras CNN and text detection together obtained:

The bounding boxes for given Image and corresponding letters associated with it



**Chapter 7**

**Conclusion:**

This project presented an enhanced MSER based scene text method. It is capable of differentiating the text part from the natural scene image and can recognize the text from the selected text region using Convulutional Neural Networks(CNN). To overcome the complexity of image blur and small letter, enhanced MSER had been developed with complementary properties of MSER. We proposed a unique image operator to determine accurately the stroke width of binary connected components. Finally, OCR with intersecting character description had been applied to the selected text part to recognize text. Our proposed system exhibits good performance over CHAR74K character dataset.

**Chapter 8**

**Limitations and Future Scope**

**8.1 Limitations:**

One of the primary limitations of this model is the small amount of training data. We have tested on various images and at times the classifications were incorrect. When this happened, We examined the input image + network more closely and found that the color(s) most dominant in the image influence the classification dramatically.

As we are using MSER for detecting the text, If the image is not of good quality or if there are any dominant colours in the background the Prediction accuracy may decrease or even falls to sharper depths.

If the image consists of any Punctuation marks then the model will not be able to predict the given punctuation in the image because our Dataset doesn’t contain the punctuation marks, as it only contain Alplanumeric values.

**8.2 Future Scope:**

Through the years, the methods of character recognition has improved from quite primitive schemes, suitable only for reading stylized printed numerals, to more complex and sophisticated techniques for the recognition of a great variety of typeset fonts and also handprinted characters. Below the future of OCR when it comes to both research and areas of applications, is briefly discussed.

**8.2.1** **Future improvements:**

New methods for character recognition are still expected to appear, as the computer technology develops and decreasing computational restrictions open up for new approaches. There might for instance be a potential in performing character recognition directly on grey level images. However, the greatest potential seems to lie within the exploitation of existing methods, by mixing methodologies and making more use of context. Integration of segmentation and contextual analysis can improve recognition of joined and split characters. Here in this system we are taking the bounding boxes obtained by the MSER and feeding it to our trained CNN model, but the future improvement of this system can be (1) Expanding the bounding boxes so that it recognises the whole word (2) and then we need to recognise each character by using another CNN which can recognise the splits between the characters in the word.

**8.2.2** **Future needs:**

Today optical character recognition is most successful for constrained material, that is documents produced under some control. However, in the future it seems that the need for constrained OCR will be decreasing. The reason for this is that control of the production process usually means that the document is produced from material already stored on a computer. Hence, if a computer readable version is already available, this means that data may be exchanged electronically or printed in a more computer readable form, for instance barcodes.

The applications for future OCR-systems lie in the recognition of documents where control over the production process is impossible. This may be material where the recipient is cut off from an electronic version and has no control of the production process or older material which at production time could not be generated electronically. This means that future OCR-systems intended for reading printed text must be omnifont. Another important area for OCR is the recognition of manually produced documents. Within postal applications for instance, OCR must focus on reading of addresses on mail produced by people without access to computer technology. Already, it is not unusual for companies etc., with access to computer technology to mark mail with barcodes. The relative importance of handwritten text recognition is therefore expected to increase.

**Bibiliography**

[1] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. High performance neural networks for visual object classification. Technical Report IDSIA-01-11, Dalle Molle Institute for Artificial Intelligence, 2011.

[2] A. Coates, B. Carpenter, C. Case, S. Satheesh,B. Suresh, T. Wang, D. J. Wu, and A. Y. Ng. Text detection and character recognition in scene images withunsupervised feature learning. In ICDAR, 2011.

[3] A. Coates, H. Lee, and A. Y. Ng. An analysis of singlelayer networks in unsupervised feature learning. In AISTATS, 2011.

[4] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In VISAPP, 2009.

[5] B. Epshtein, E. Oyek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. In CVPR,2010.

[6] A. Hyvarinen and E. Oja. Independent component analysis: algorithms and applications. Neural networks, 13(4-5):411–430, 2000.

[7] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng. Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In CVPR, 2011.

[8] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E.Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. Neural Computation, 1:541–551, 1989.

[9] C.-L. Liu, M. Koga, and H. Fujisawa. Lexicon-driven segmentation and recognition of handwritten character strings for japanese address reading. IEEE Trans. Pattern Anal. Mach. Intell., 24(11):1425–1437, Nov. 2002.