1. INTRODUCTION

Text recognition in natural image has received increasing attention in computer vision and machine intelligence, due to its numerous practical applications like text extraction using optical character recognition, reflowing documents, and layout-based document retrieval. This problem includes two sub tasks, namely text detection and text-line/word recognition. Text detection in document images is performed by using segmentation which separates text and graphics regions.

Various steps involved in text detection and recognition process are Image Inversion, Binarization, Skew Detection and correction, Smearing, Connected Component Analysis, Noise reduction, and classification of text and graphics regions.

On performing Page segmentation, a scanned document image is divided into columns and blocks which are then classified as graphics, or text based on some properties like width, height and area of the connected components. Segmentation algorithms are divided into four categories: bottom-up, top-down, hybrid and multi-scale resolution. In this project, the Run Length Smearing algorithm (RLSA) is implemented for page segmentation, a top-down based segmentation algorithm.

Segmentation can be division of document images in to text and graphics regions called as page segmentation, division of page into lines called as line segmentation, division of page into words called as word segmentation.

On performing segmentation on Telugu document images containing both text and graphics, the segmentation algorithm separates the text and graphics regions. The next step is to convert the binarized text regions into word images which are used for training and text recognition by using a convolutional neural network.

1.1 Existing system

Until recently, feature engineering was the dominant approach that used features like Wavelet features, Skelton features etc., followed by SVM (Support Vector Machine) or boosting based classifiers. Most of these works on character recognition are based on different languages and these mostly worked on identifying characters.

1.2 Proposed System

The success of CNN (Convolutional Neural Networks) motivated us to use them for Telugu character recognition. The first stage is skew correction in which tilt in the image will be adjusted, followed by binarization and word segmentation after which each document is converted into separate words and binarized. The classifier engine is created by training on a dataset of the word images along with their labels.

1.3 Requirements Specification

Hardware Requirements:

- It requires a minimum of 2.16 GHz processor.
- It requires a minimum of 4 GB RAM.
- It requires 64-bit architecture
- It requires a minimum storage of 500GB

Software Requirements:

- It requires a 64-bit Ubuntu Operating System.
- Python language for coding.
- Libraries – numpy, keras, matplotlib.plt.

User requirements and system requirements may be defined as follows:

- User requirements are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.
- System requirements are more detailed descriptions of the software system's functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.

User Requirement Definition

Telugu word Recognition using CNN system shall be able to perform segmentation and Telugu word recognition.

System Requirements Specification

- The system shall perform binarization on the input image.
- Next, Deskewing and smearing are to be performed on that image which forms connected components.
- By using the properties of the connected components, the system separates the word and graphic regions into separate folders.
- The word images thus separated from the document image are used for training and recognition by using a Convolutional neural network.

2. LITERATURE SURVEY

Tuan Anh Tran, Kanghan Oh, In-Seop Na, 2017 proposed a robust document layout analysis system for which the homogeneity structure is used as a basis structure, which contains three stages: First the text and non-text elements of the document image are classified by the use of multilevel and multi-layer homogeneity structure. Second, the text elements in the text document are grouped and segmented based on the combination of text line extraction, paragraph segmentation. Third, all of the regions are refined to remove he noise and enhance the quality of each region.

Saad ALBAWI, Tareq Abed MOHAMMED, Saad AL-ZAWI, 2017 explained and defined all the elements and important issues related to CNN, and how these elements work. In addition, they have also stated the parameters that effect CNN efficiency.

Scott Leishman, 2007 presented a technique for improving the robustness of optical character recognition, by focusing fundamentally on exploiting contextual cues and linguistic properties to determine the mapping from blobs of ink to underlying character symbols. This is largely motivated by human behavior. For a language like English, Humans are readily aware that certain sequences of characters are much more likely than others and know that just about every word will contain at least one vowel character, even that the ordering of words must follow a grammatical structure. Instead of relying on shape information to do the heavy lifting during the recognition process (with context being used as an afterthought to fill in the gaps and improve low confidence mappings), Working in a top-down fashion in which completely shape information during the recognition process is ignored, instead relying on contextual clues to do much of the work in determining character identities. It is hoped that by operating in this manner, the recognition process will automatically adapt to the specific nuances and constraints of each input document. Such a system becomes entirely independent of character font and shape, and with this variation removed, improved robustness should result.

The work by **Rakesh and Trevor**, 2015 on Telugu OCR using convolutional neural networks is also fascinating. They used 50 fonts in four styles for training data each image of size 48x48. However, they not consider all possible outputs (only 457classes) of CNN.

The work by **Kunte and Samuel**, 2007 on Kannada OCR employs a 2 stage classification system that is similar to our approach. They have first used wavelets for feature extraction and then two stage multi-layer perceptrons for the task of classification. They have divided the characters into separate sub classes but have not considered all possible combinations.

The work by **Jawahar et.al**, 2003 describes a bilingual Hindi-Telugu OCR for documents containing Hindi and Telugu text. It is based on Principal Component analysis followed by support vector regression. They report an overall accuracy of 96.7% over an independent test set. They perform character level segmentation offline by their data collecting tools. However, they have only considered 330 distinct classes.

The first attempt to use neural networks was made by **M.B. Sukhaswami** et.al, 1995 which trains multiple neural networks, pre-classify an image based on its aspect ratio and feed it to the corresponding network. It demonstrated the robustness of a Hopfield network for the purpose of recognition of noisy Telugu characters. Later work on Telugu OCR primarily followed the featurization classification paradigm.

S.NO	Name of the Author	Year	Name of the paper	Description
1	Tuan Ann Tran, Kanghan Oh, In-Seop Na, Guee-Sang Lee, Hyung-Jeong Yang, Soo-Hyung Kim.	2017	A robust system for document layout analysis using multilevel homogeneity structure	A robust document layout analysis system for which the homogeneity structure is used as a basis structure.
2	Saad ALBAWI , Tareq Abed MOHAMME Saad ALZAWI	2017	Understanding of a Convolutional Neural Network	Explain and define all the elements and important issues related to CNN, and how these elements work.
3	Faisal Shafait	2011	Document and Content Analysis	Explained the importance of binarization technique for Optical Character Recognition and explained local and global thresholding techniques.
4	Scott Leishman	2007	Shape-Free Statistical Information in Optical Character Recognition	Explained about connected component analysis: 4-connected component analysis and 8-connected component analysis.
5	Jawahar, C. V., MNSSK Pavan Kumar, and SS Ravi Kiran	2003	A bilingual OCR for Hindi-Telugu documents and its applications	It is based on Principal Component analysis followed by support vector regression.

3. DESIGN OF TELUGU WORD RECOGNITION USING CNN

3.1 System Architecture

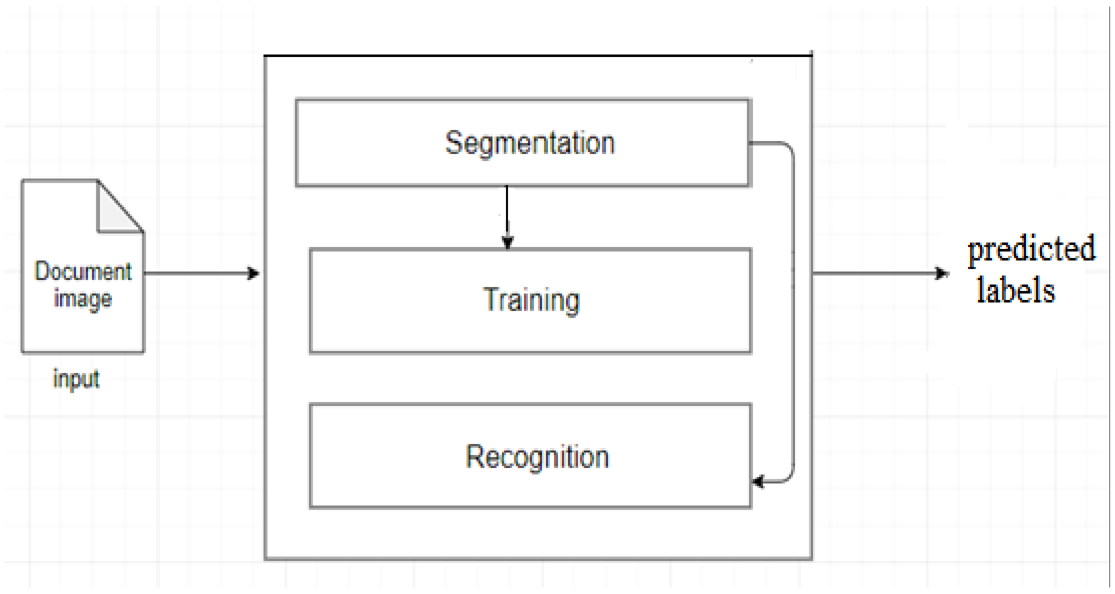


Figure 3.1: System Architecture of Telugu word Recognition using CNN

Telugu word Recognition using CNN System takes a Telugu document image containing text and non-text regions as input. In segmentation process the input image is segmented into words and images. The obtained word mages are then used for training and testing. The training set consists of 4500 word images taken from four books of around hundred pages each.

3.2 Modules

Telugu word Recognition using CNN consists of two modules:

1. Page Segmentation and

2. Telugu word Recognition.

Page segmentation:

Consists of following modules:

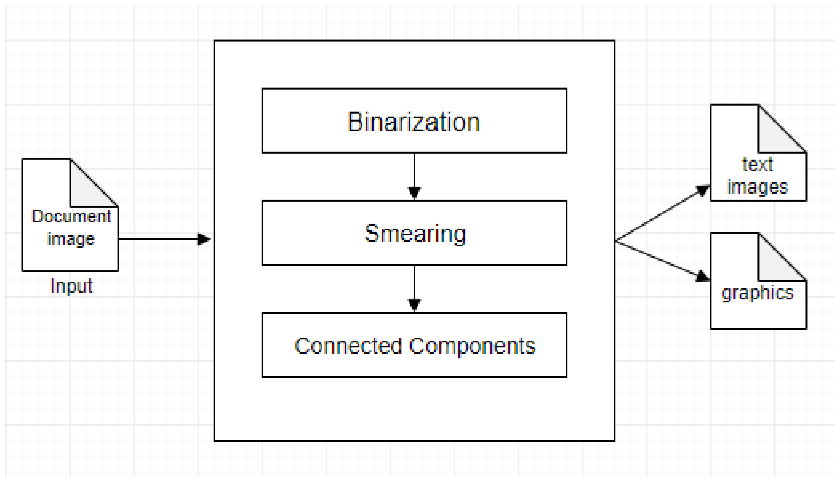


Figure 3.2: Diagram of Page Segmentation Process.

Algorithm: Word Segmentation

Procedure WordSegmenattion(DocImage)

Binarize:

```
For all Pixels do
    If pixelvalue < 127
        Pixelvalue <- 0
    Else
        Pixelvalue = 255
```

Smearing: white pixels are represented by 0's and black pixels by 255's.

For each row

1. 0's in x are changed to 255's in y if the number of adjacent 0's is less than or equal to a predefined limit C (Threshold).

2. 255's in x are unchanged in y.

Connected Component analysis:

With connectivity 8, find the connected components and their properties by using

```
cv2.connectedComponentsWithStats function
With the help of properties width and height compute the area of components
If the area of components < threshold
    Classify it as noise and ignore it
Else:
    If the height < 80 (for text size)
        Classify it as word image
    Else
        Classify it as graphics image
```

3.2.1 Binarization

For character recognition purposes, one generally does not need a full colour representation of the image, and so pages are often scanned or converted to a grayscale or bilevel colour depth. In grayscale, each pixel represents one of 256 shades of gray, and in a bilevel image each pixel is assigned one of two values representing black or white. Working with bilevel images is particularly efficient, and some processing algorithms are restricted to this format.

The process of converting a colour or grayscale image to bilevel format is referred to as **binarization**. Several approaches to binarization have been discussed in the literature but they typically fall into one of two categories[3]. Global methods treat each pixel independently, converting each to black or white based on a single threshold value. If a pixel's colour intensity is higher than the global threshold it is assigned one value, otherwise it is assigned the opposite value. In contrast local methods, make use of the colour information in nearby pixels to determine an appropriate threshold for a particular pixel.

3.2.2 Smearing

Run Length Smoothing Algorithm (RLSA):

The Run Length Smoothing Algorithm (RLSA) is a method that can be used for Block segmentation and text discrimination[5]. The method developed for the Document Analysis System consists of two steps. First, a segmentation procedure subdivides the area of a document into regions (blocks), each of which should contain only one type of data (text, graphic, halftone image, etc.). Next, some basic features of these blocks are calculated such as height, width and area[4].

The basic RLSA is applied to a binary sequence in which white pixels are represented by 0's and black pixels by 1's. The algorithm transforms a binary sequence x into an output sequence y according to the following rules:

1. 0's in x are changed to 1's in y if the number of adjacent 0's is less than or equal to a predefined limit C (Threshold).

2. 1's in x are unchanged in y.

For example, with C = 4 the sequence x is mapped into y as follows:

x : 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0

y : 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1

When applied to pattern arrays, the RLSA has the effect of linking together neighboring black areas that are separated by less than C pixels. With an appropriate choice of C, the linked areas will be regions of a common data type.

The RLSA is applied row-by-row as well as column-by-column to a document, yielding two distinct bit-maps. Because spacings of document components tend to differ horizontally and vertically, different values of C are used for row and column processing.

3.2.3 Connected Components Analysis

A common approach to identifying isolated symbol images, begins by grouping together touching (connected) pixels to create a set of connected components[2]. Given a binary input page P, individual pixels can be represented by $p(x,y) = v$ where x denotes the horizontal and y the vertical distance (measured as the number of pixels) away from the origin of the page. Often, this origin is set to the top-left corner of the page.

The pixel value v for binary intensity images implies that v must be one of 0, or 1 with the convention (used throughout the remainder of this thesis) that foreground pixels will have $v = 1$, and background pixels $v = 0$. Using this description then, a foreground pixel $p(x,y)$ is said to be connected to a second foreground pixel $p(x_0,y_0)$ if and only if there is a sequence of neighbouring foreground pixels $p(x_1,y_1),...p(x_n,y_n)$ where $x = x_1$, $y = y_1$ and $x_0 = x_n, y_0 = y_n$. Two pixels $p(x_i,y_i)$ and $p(x_j,y_j)$ are said to be neighbouring under what is called an 8-connected scheme exactly when $|x_j - x_i| \leq 1$ and $|y_j - y_i| \leq 1$. The same two pixels are considered neighbouring under a 4-connected scheme when one of the following two cases holds: either $|x_j - x_i| \leq 1$ and $|y_j - y_i| = 0$, or $|x_j - x_i| = 0$ and $|y_j - y_i \leq 1$. For 4-connected schemes, neighbouring pixels must be vertically or horizontally adjacent, whereas in 8-connected schemes, neighbouring pixels can be vertically, horizontally, or diagonally adjacent.

Both of these schemes are illustrated in Figure 3.3. A single connected component ci is then defined as a set of foreground pixels such that each is pairwise connected with each other pixel in that set.

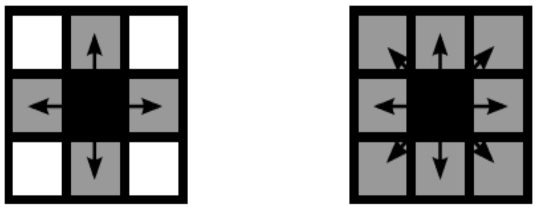


Figure 3.3: A single pixel and its 4-connected neighborhood (left), and 8-connected neighborhood (right).

You need to choose 4 or 8 for connectivity type

```
Connectivity=8
output = cv2.connectedComponentsWithStats(thresh, connectivity, cv2.CV_32S)
# Get the results
# The first cell is the number of labels
num_labels = output [0]
# The second cell is the label matrix
labels = output [1]
# The third cell is the stat matrix
stats = output [2]
# The fourth cell is the centroid matrix
centroids = output[3]
```

Labels is a matrix of the size of the input image where each element has a value equal

to its label.

Stats is a matrix of the stats that the function calculates. It has a length equal to the number of labels and a width equal to the number of stats. It can be used with the OpenCV documentation for it:

Statistics are accessed via stats [label, COLUMN] where available columns are defined below.

- cv2.CC_STAT_LEFT The leftmost (x) coordinate which is the inclusive start of the bounding box in the horizontal direction.
- cv2.CC_STAT_TOP The topmost (y) coordinate which is the inclusive start of the bounding box in the vertical direction.
- cv2.CC_STAT_WIDTH The horizontal size of the bounding box
- cv2.CC_STAT_HEIGHT The vertical size of the bounding box
- cv2.CC_STAT_AREA The total area (in pixels) of the connected component

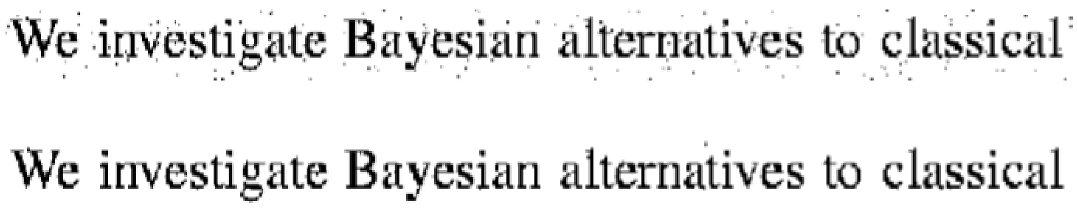
Centroids is a matrix with the x and y locations of each centroid. The row in this matrix corresponds to the label number.

3.2.4 Noise Removal

During the scanning process, differences between the digital image and the original input (beyond those due to quantization when stored on computer) can occur. Such unwanted marks and differing pixel values constitute noise that can potentially skew character recognition accuracy.

There are typically two approaches taken to remove unwanted noise from document images. For pixels that have been assigned a foreground value when a background value should have been given (additive noise), correction can sometimes be accomplished by removing any groups of foreground pixels that are smaller than some threshold. These groups of foreground pixels can be identified by employing a connected component sweep over the document. Care must be taken to ensure that groups of pixels corresponding to small parts of characters or symbols (dots above i, or j, or small punctuation like., etc.) are left intact.

Additive noise that is directly adjacent to other foreground pixels will not be corrected by this approach, and so each instance of a character may appear slightly different on a page. Figure 3.4 illustrates this, showing an originally noisy image region, and its denoised equivalent after small blobs containing fewer than 3 pixels are removed (note that the noise around each of the characters remains).


We investigate Bayesian alternatives to classical

We investigate Bayesian alternatives to classical

Figure 3.4: Closeup of a noisy image region, and result after denoising.

Telugu word Recognition:

Consists of following modules:

3.2.5 Convolutional Neural Network (CNN)

A convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. Convolutional neural networks use a variation of multilayer perceptrons designed to require minimal preprocessing[6].

A Convolutional Neural Network can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

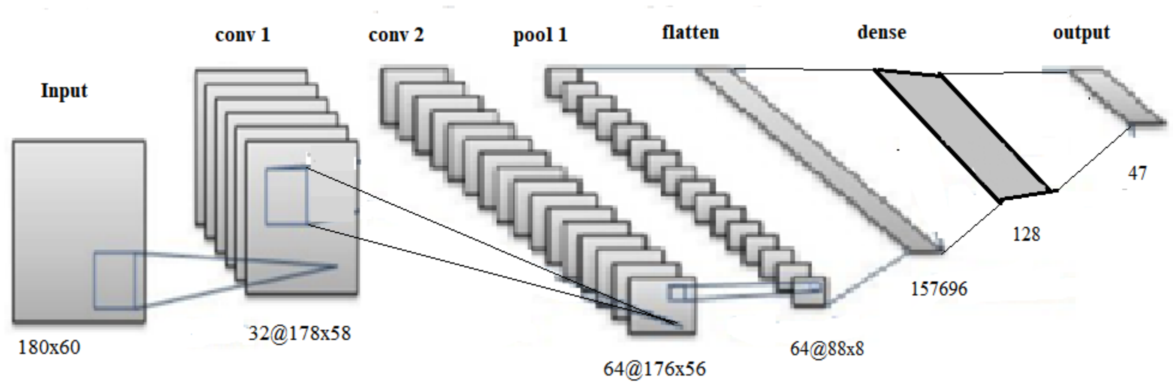
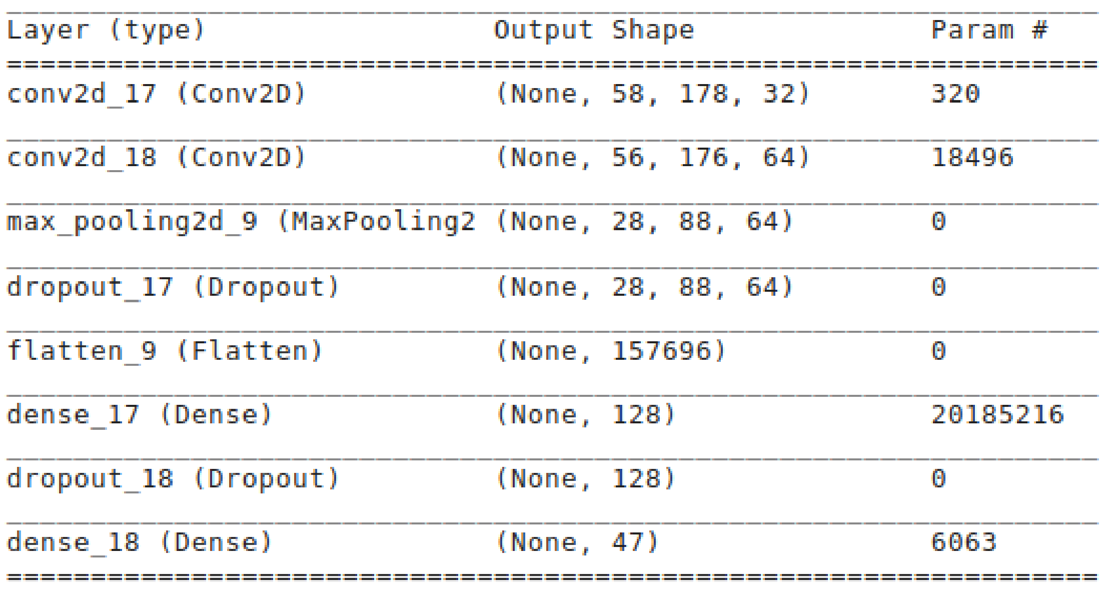


Figure 3.5: Block diagram of CNN.

The figure 3.5, shows the Convolutional neural network used in this system. It takes an image of size 180x60 as input. The first convolutional layer has 32 filters each with a size of 3x3. The Second Convolutional layer has 64 filters each with a size of 3x3. The output of this convolutional layer is input to a maxpooling layer with a pool size of 2x2. The output of pooling layer is flattened which resulting in a layer containing 157696 neurons. These are connected to a dense layer having 128 perceptrons.

These are connected to a last dense layer containing 47 perceptrons and a softmax function is employed to make the output of these neurons as probabilities.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_17 (Conv2D)           (None, 58, 178, 32)       320       
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 56, 176, 64)       18496     
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 28, 88, 64)        0         
_________________________________________________________________
dropout_17 (Dropout)         (None, 28, 88, 64)        0         
_________________________________________________________________
flatten_9 (Flatten)          (None, 157696)            0         
_________________________________________________________________
dense_17 (Dense)             (None, 128)               20185216  
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_18 (Dense)             (None, 47)                6063      
=================================================================
```

Figure 3.6: Convolutional neural network model summary

The above figure 3.6, shows the summary of CNN model used in this system which describes the output shape. For example the output of first layer consists of 58 images each of size 178x32.

Convolutional networks were inspired by biological processes[7] in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers[6]. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and RELU activation function.

3.2.6 Convolution (CONV)

The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height and 3 because images have depth 3, the color channels). During the forward pass, each filter is slided (more precisely, convolve) across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As the filter is slided over the width and height of the input volume, it will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, CNN will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. These activation maps are stacked to produce the output volume.

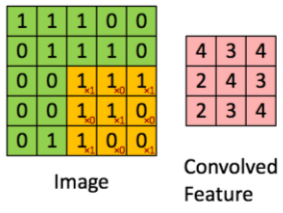


Figure 3.7: Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature.

In the figure 3.7, the green section resembles our 5x5x1 input image. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the color yellow. We have selected K as a 3x3x1 matrix

Kernel/Filter, K = 1 0 1

0 1 0

1 0 1

3.2.7 Pooling

Convolutional networks may include local or global pooling layers. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer[7]. In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer.

The Convolutional Layer and the Pooling Layer, together form the i^{th} layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

3.2.8 Fully-connected layer (Dense)

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

Convolution layer with a size of 64@28x28 gives rise to a fully connected later with 157696 neurons. 64@28x28 represents that 64 filters applied an image size of 28x28 in the layer.

3.2.9 Activation Function

Activation function is a node that is put at the end of or in between Neural Networks. They help to decide if the neuron would fire or not. ReLU (Rectified Linear Unit) is linear (identity) for all positive values, and zero for all negative values.

The different types of activation functions are Sigmoid, tanh, Leaky ReLU and Rectified Linear Unit (ReLU).

ReLU function is the most widely used activation function in neural networks today. One of the greatest advantage ReLU has over other activation functions is that it does not activate all neurons at the same time.

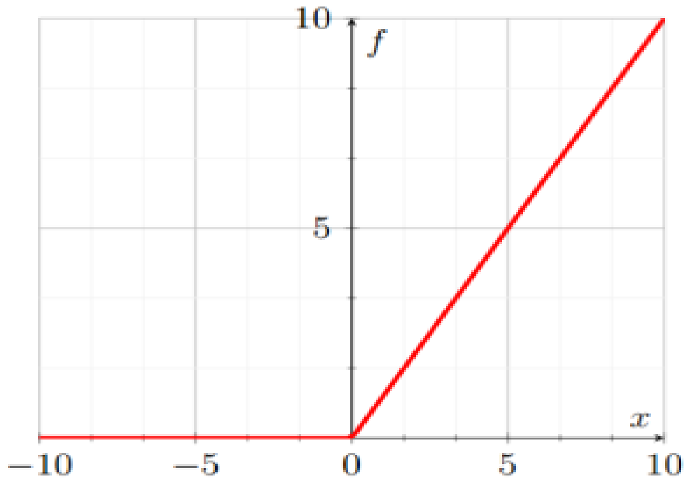


Figure 3.8 Rectified Linear Unit (ReLU)

From the image for ReLU function above, it converts all negative inputs to zero and the neuron does not get activated. It's surprising that such a simple function (and one composed of two linear pieces) can allow your model to account for non-linarites and interactions so well. But the ReLU function works great in most applications, and it is very widely used as a result. This makes it very computational efficient as few neurons are activated per time. It does not saturate at the positive region. In practice, ReLU converges six times faster than tanh and sigmoid activation functions.

3.2.10 Softmax function

The softmax function, also known as softargmax or normalized exponential function, is a function that takes as input a vector of *K* real numbers, and normalizes it into a probability distribution consisting of *K* probabilities. That is, prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying the softmax, each component will be in the interval (0, 1), and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities. Softmax is often used in neural_networks, to map the non-normalized output of a network to a probability distribution over predicted output classes.

The standard (unit) softmax function $\sigma : Rk \rightarrow Rk$ is defined by the formula

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

In words: applying the standard exponential function to each element z_i of the input vector z and normalize these values by dividing by the sum of all these exponentials this normalization ensures that the sum of the components of the output vector $\boldsymbol{\sigma}$(z) is 1.

3.3 UML DIAGRAMS

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.

3.3.1 Use Case Diagram:

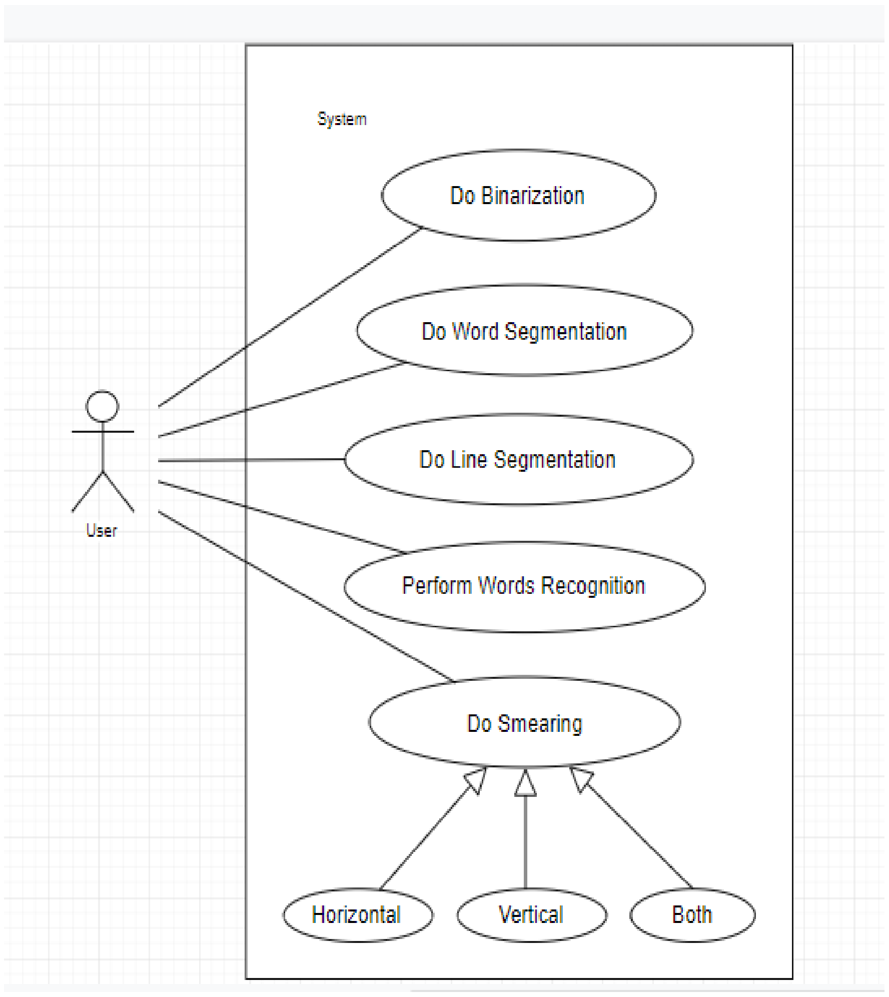


Figure 3.9: Use case diagram for Telugu word Recognition using CNN system

The above figure 3.9, shows the use case diagram of the system. It has one actor who is the user of the system. The user can perform the binarization, smearing in horizontal, vertical and combination of both, word segmentation, line segmentation and also perform the words recognition using Convolutional Neural Network.

3.3.2 Activity Diagram

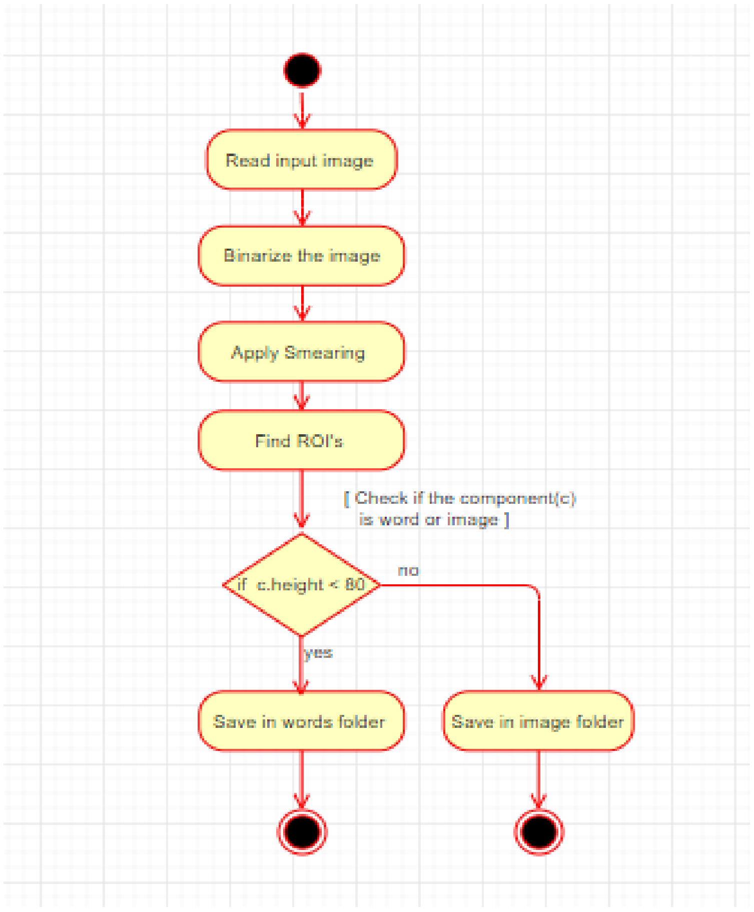


Figure 3.10: Activity diagram for Telugu words Recognition using CNN system.

As shown in the figure 3.10, system takes the Telugu document image and performs the binarization, smearing and finding the connected components. After finding the connected components, the height of the connected components are tested to determine whether the connected component is text or images and stored in separate folders.

3.3.3 Component Diagram

Component diagram does not describe the functionality of the system but it describes the components used to make those functionalities.

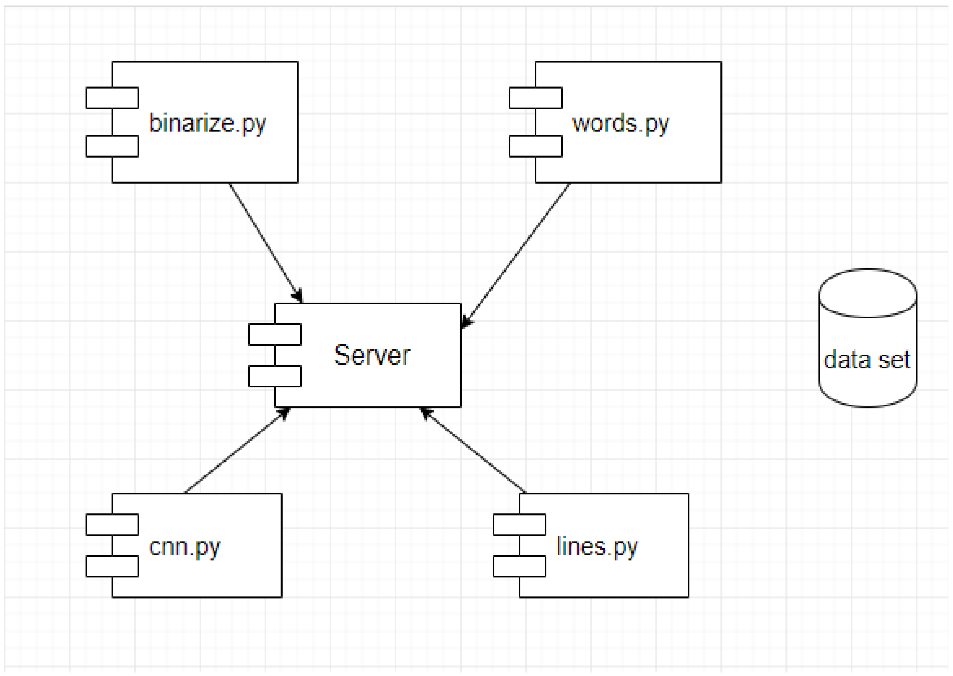


Figure 3.11: Component diagram for page segmentation and Telugu words recognition system.

The above figure 3.11, shows the various components are binarize.py, words.py, cnn.py and lines.py. Binarize.py component takes the grayscale image and converted into binarized image. The words.py component will separates the words and images from the binarized image and as well as lines.py separates the lines instead of words. The cnn.py component perform the training and testing on the given dataset and returns the results.

4. TESTING AND RESULTS

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product.

It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTING

Unit Testing

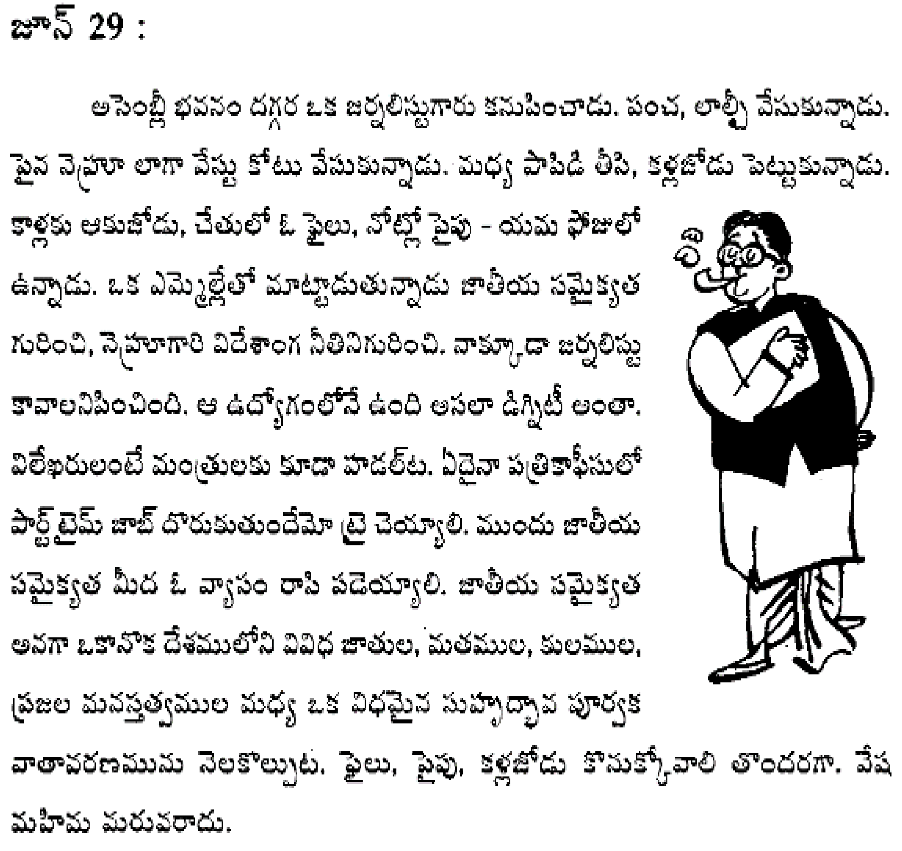
Unit testing focuses verification effort on the smallest unit of software design—the software component or module. It is done after the completion of an individual unit before integration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Each module is tested separately by invoking each module separately from the command prompt and checked for appropriate results. Debug statements are inserted and checked for unit testing of each part of the function. Each activity is run separately to verify the function.

Integration Testing

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design.

The project is thoroughly tested by testing the each and every component and verifying the results.



జూన్ 29 :

అసెంబ్లీ భవనం దగ్గర ఒక జర్నలిస్టుగారు కనుపించాడు. పంచ, లాల్చీ వేసుకున్నాడు.
పైన నెహ్రూ లాగా వేస్టు కోటు వేసుకున్నాడు. మధ్య పాపిడి తీసి, కళ్లజోడు పెట్టుకున్నాడు.
కాళ్లకు అకుజోడు, చేతులో ఓ ఫైలు, నోట్లో పైపు - యమ ఫోజులో
ఉన్నాడు. ఒక ఎమ్మెల్లేతో మాట్లాడుతున్నాడు జాతీయ సమైక్యత
గురించి, నెహ్రూగారి విదేశాంగ నీతినిగురించి. నాక్కూడా జర్నలిస్టు
కావాలనిపించింది. ఆ ఉద్యోగంలోనే ఉంది అసలా డిగ్నిటీ అంతా.
విలేఖరులంటే మంత్రులకు కూడా హడల్‌ట. ఏదైనా పత్రికాఫీసులో
పార్ట్ టైమ్ జాబ్ దొరుకుతుందేమో ట్రై చెయ్యాలి. ముందు జాతీయ
సమైక్యత మీద ఓ వ్యాసం రాసి పడెయ్యాలి. జాతీయ సమైక్యత
అనగా ఒకానొక దేశములోని వివిధ జాతుల, మతముల, కులముల,
ప్రజల మనస్తత్వముల మధ్య ఒక విధమైన సుహృద్భావ పూర్వక
వాతావరణమును నెలకొల్పుట. ఫైలు, పైపు, కళ్లజోడు కొనుక్కోవాలి తొందరగా. వేష
మహిమ మరువరాదు.

Figure 4.1: Sample telugu document image.

The above figure 4.1, shows the input image to the system. If the input image is grayscale, each pixel represents one of 256 shades of gray, and in a bilevel image each pixel is assigned one of two values representing black or white. Working with bilevel images is particularly efficient, for that the system perform the binarization to convert the grayscale image into bilevel image.

4.1 TEST CASES

Test case 1: The first module tested is horizontal smearing module which takes the input image and perform the horizontal smearing with threshold value 35. This module is tested by performing unit testing by executing the command python horizontal.py in the command prompt. The figure 4.2, shows the output of the horizontal smearing module.

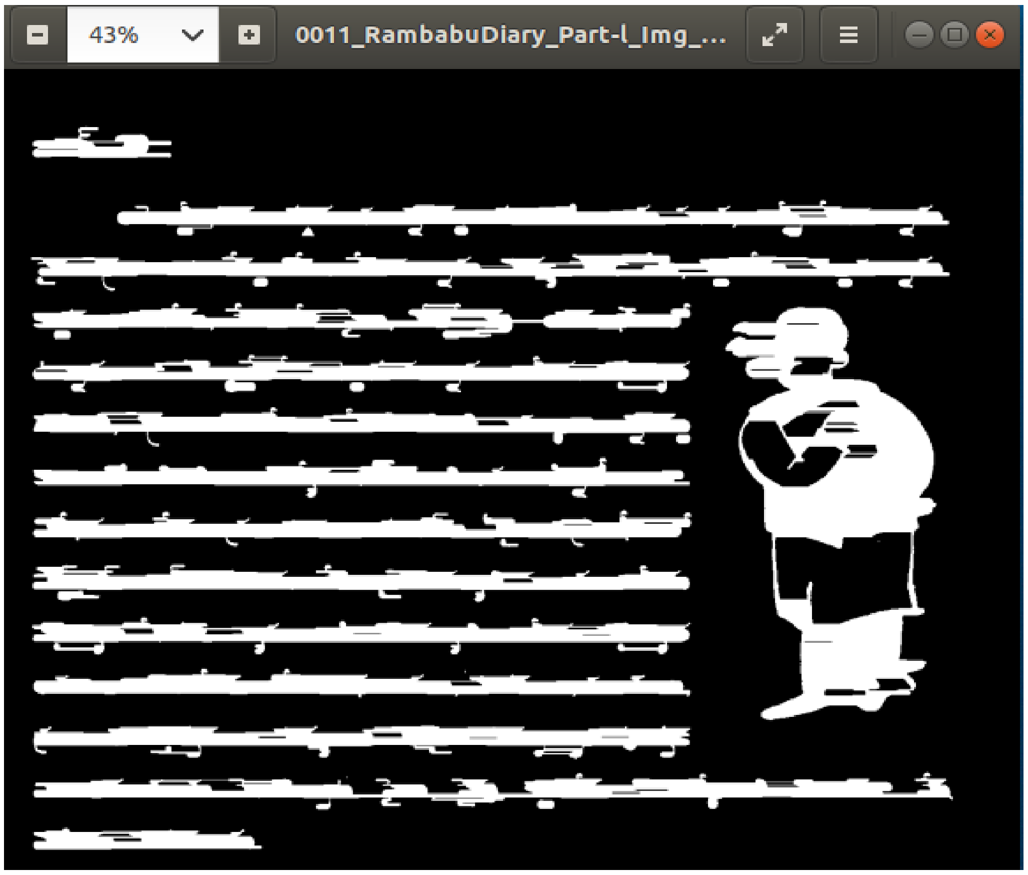


Figure 4.2: Output of horizontal smearing.

Test case 2: The vertical smearing module is tested by performing the unit testing by entering the command python vertical.py. It performs the smearing in vertical direction with threshold value 20 and figure 4.3, shows the output of the vertical smearing module.

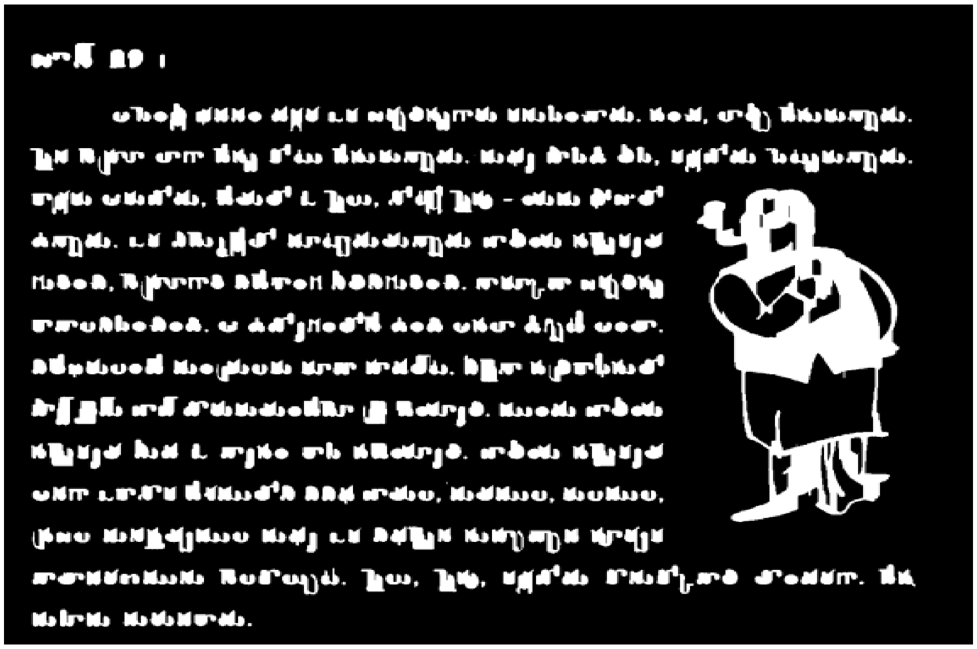


Figure 4.3: output of vertical smearing.

Test case 3: the Combination of horizontal and vertical smearing module is tested by entering the command python both.py. This module is performs both horizontal and vertical smearing to form the components. Figure 4.4, shows the output of this module.

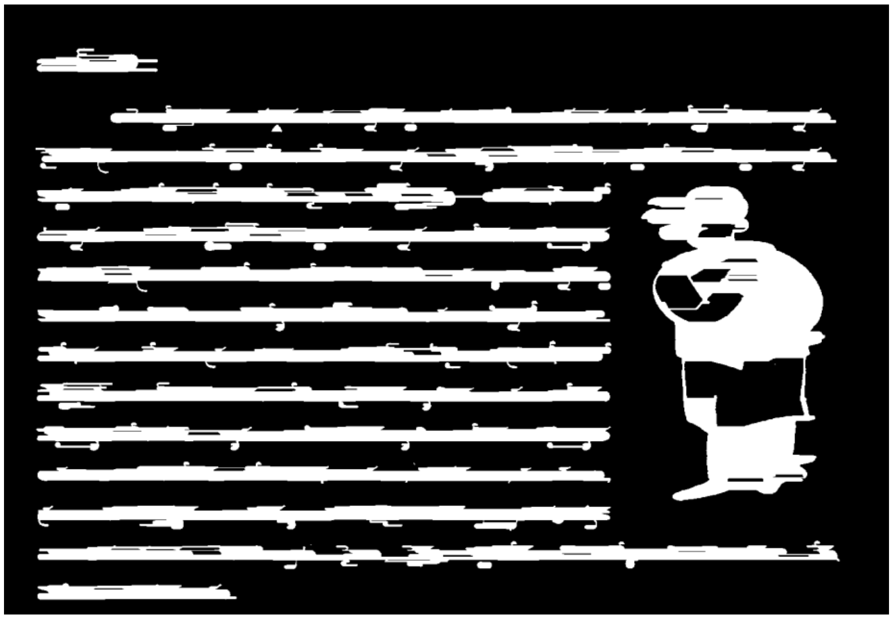


Figure 4.4 output of both horizontal and vertical smearing.

Test case 4: line segmentation is tested by executing the command python lines.py in the command prompt. This module separates the text (lines) and non-text regions and stored in separate folders. Figure 4.5, shows the output of line segmentation module.

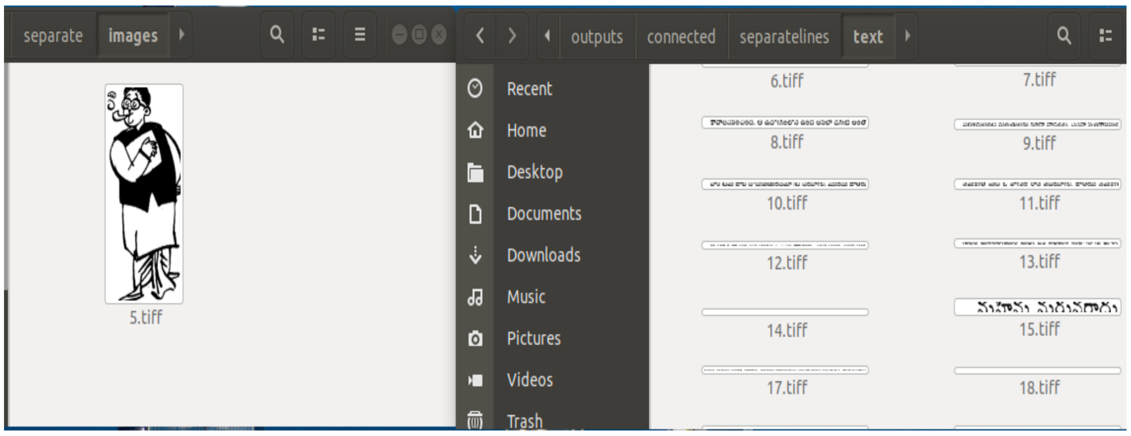


Figure 4.5: output of line segmentation.

Test case 5: Word segmentation is tested by executing the command python words.py in the command prompt. This module separates the text (words) and images form the input image and stored in the separate folders. Figure 4.6, shows the output of the word segmentation module.

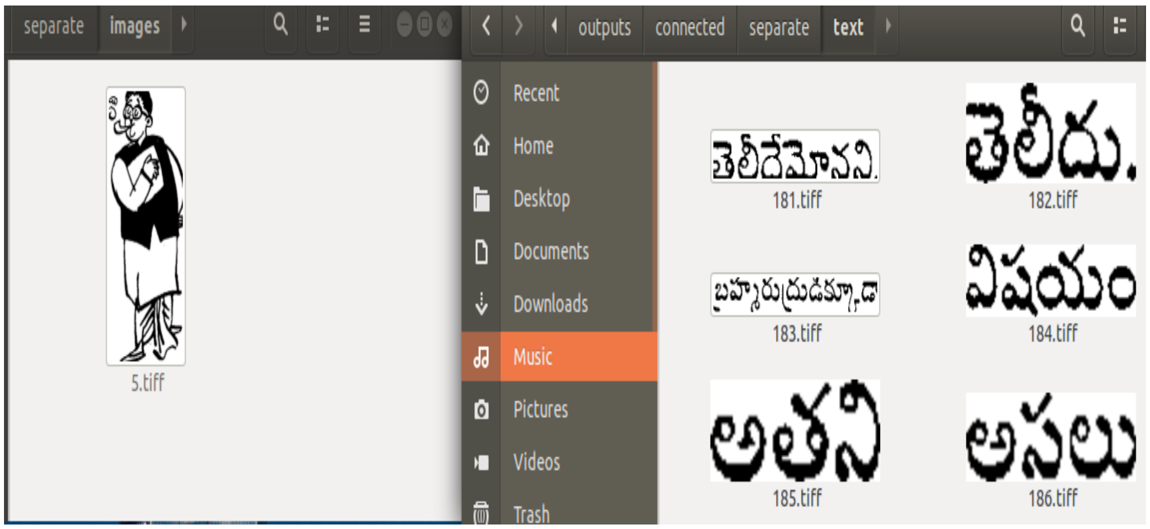
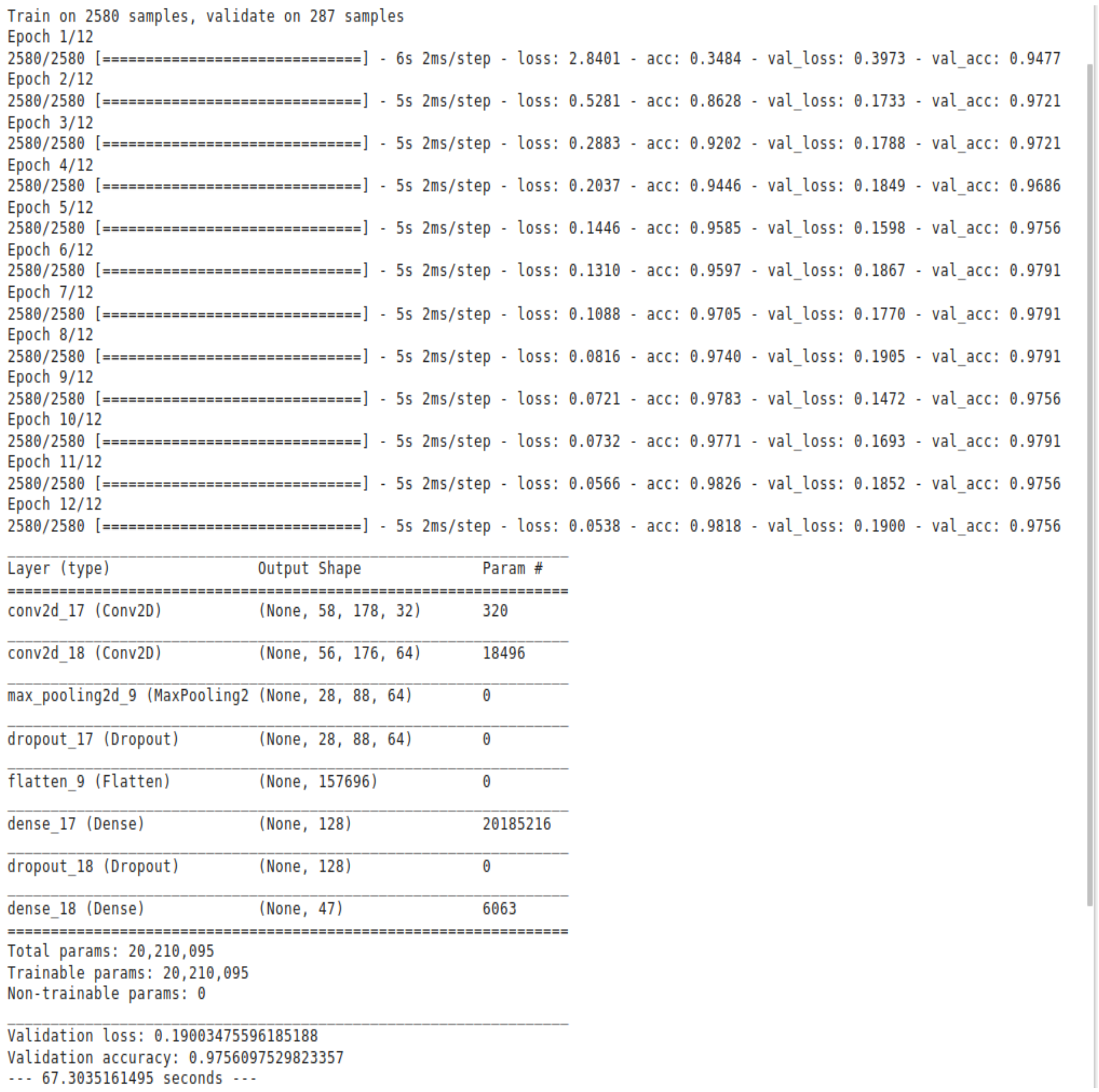


Figure 4.6: output of word segmentation.

Test case 6: Convolution Neural Network for word recognition is tested by executing the program by entering the command cnn.py in the command prompt.



```
Train on 2580 samples, validate on 287 samples
Epoch 1/12
2580/2580 [==============================] - 6s 2ms/step - loss: 2.8401 - acc: 0.3484 - val_loss: 0.3973 - val_acc: 0.9477
Epoch 2/12
2580/2580 [==============================] - 5s 2ms/step - loss: 0.5281 - acc: 0.8628 - val_loss: 0.1733 - val_acc: 0.9721
Epoch 3/12
2580/2580 [==============================] - 5s 2ms/step - loss: 0.2883 - acc: 0.9202 - val_loss: 0.1788 - val_acc: 0.9721
Epoch 4/12
2580/2580 [==============================] - 5s 2ms/step - loss: 0.2037 - acc: 0.9446 - val_loss: 0.1849 - val_acc: 0.9686
Epoch 5/12
2580/2580 [==============================] - 5s 2ms/step - loss: 0.1446 - acc: 0.9585 - val_loss: 0.1598 - val_acc: 0.9756
Epoch 6/12
2580/2580 [==============================] - 5s 2ms/step - loss: 0.1310 - acc: 0.9597 - val_loss: 0.1867 - val_acc: 0.9791
Epoch 7/12
2580/2580 [==============================] - 5s 2ms/step - loss: 0.1088 - acc: 0.9705 - val_loss: 0.1770 - val_acc: 0.9791
Epoch 8/12
2580/2580 [==============================] - 5s 2ms/step - loss: 0.0816 - acc: 0.9740 - val_loss: 0.1905 - val_acc: 0.9791
Epoch 9/12
2580/2580 [==============================] - 5s 2ms/step - loss: 0.0721 - acc: 0.9783 - val_loss: 0.1472 - val_acc: 0.9756
Epoch 10/12
2580/2580 [==============================] - 5s 2ms/step - loss: 0.0732 - acc: 0.9771 - val_loss: 0.1693 - val_acc: 0.9791
Epoch 11/12
2580/2580 [==============================] - 5s 2ms/step - loss: 0.0566 - acc: 0.9826 - val_loss: 0.1852 - val_acc: 0.9756
Epoch 12/12
2580/2580 [==============================] - 5s 2ms/step - loss: 0.0538 - acc: 0.9818 - val_loss: 0.1900 - val_acc: 0.9756
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 58, 178, 32)       320
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 56, 176, 64)       18496
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 28, 88, 64)        0
_________________________________________________________________
dropout_17 (Dropout)         (None, 28, 88, 64)        0
_________________________________________________________________
flatten_9 (Flatten)          (None, 157696)            0
_________________________________________________________________
dense_17 (Dense)             (None, 128)               20185216
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_18 (Dense)             (None, 47)                6063
=================================================================
Total params: 20,210,095
Trainable params: 20,210,095
Non-trainable params: 0
_________________________________________________________________
Validation loss: 0.19003475596185188
Validation accuracy: 0.9756097529823357
--- 67.3035161495 seconds ---
```

Figure 4.7: Summary of CNN model.

The above figure 4.7, shows the summary of the convolutional neural network. It is trained by 2580 training samples and validated on 287 samples. By executing the cnn.py program using 47 labels with 12 epochs, it gets the accuracy is 97.5%.

4.2 OUTPUT SCREENS

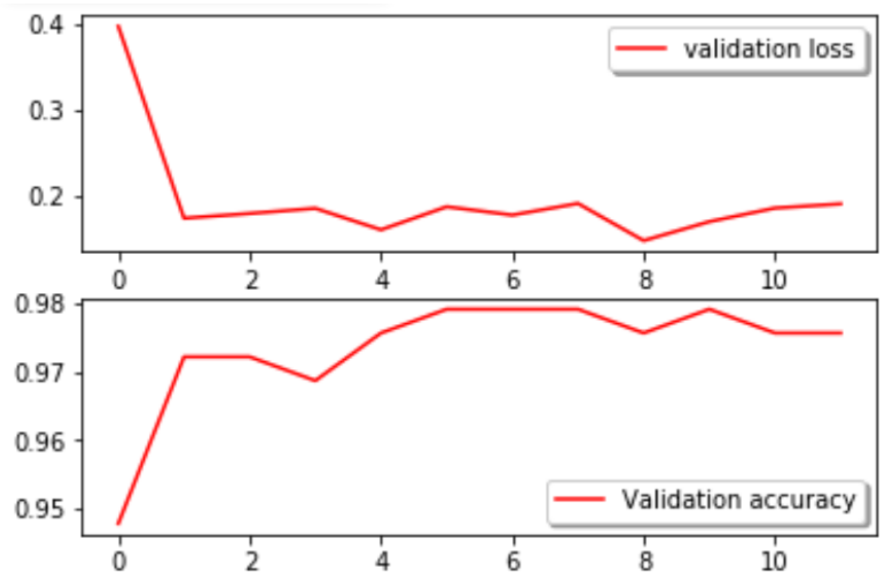


Figure 4.8: Validation loss and accuracy graphs.

The above figure 4.8, shows the output of validation loss and validation accuracy graphs. In the graph the x-axis describes the number of epochs for validation loss and accuracy and the y-axis describes the level of accuracy percentage for validation accuracy and loss percentage for validation loss. The accuracy and loss are calculated using categorical_cross_entropy function.

Each predicted probability is compared to the actual class output value (0 or 1) and a score is calculated that penalizes the probability based on the distance from the expected value. The penalty is logarithmic, offering a small score for small differences (0.1 or 0.2) and enormous score for a large difference (0.9 or 1.0). Cross-entropy loss is minimized, where smaller values represent a better model than larger values. A model that predicts perfect probabilities has a cross entropy or log loss of 0.0.

The Python function below provides a pseudocode-like working implementation of a function for calculating the cross-entropy for a list of actual one hot encoded values compared to predicted probabilities for each class.

```
from math import log
# calculate categorical cross entropy
def categorical_cross_entropy(actual, predicted):
    sum_score = 0.0
    for i in range(len(actual)):
        for j in range(len(actual[i])):
            sum_score += actual[i][j] * log(1e-15 + predicted[i][j])
    mean_sum_score = 1.0 / len(actual) * sum_score
    return mean_sum_score
```

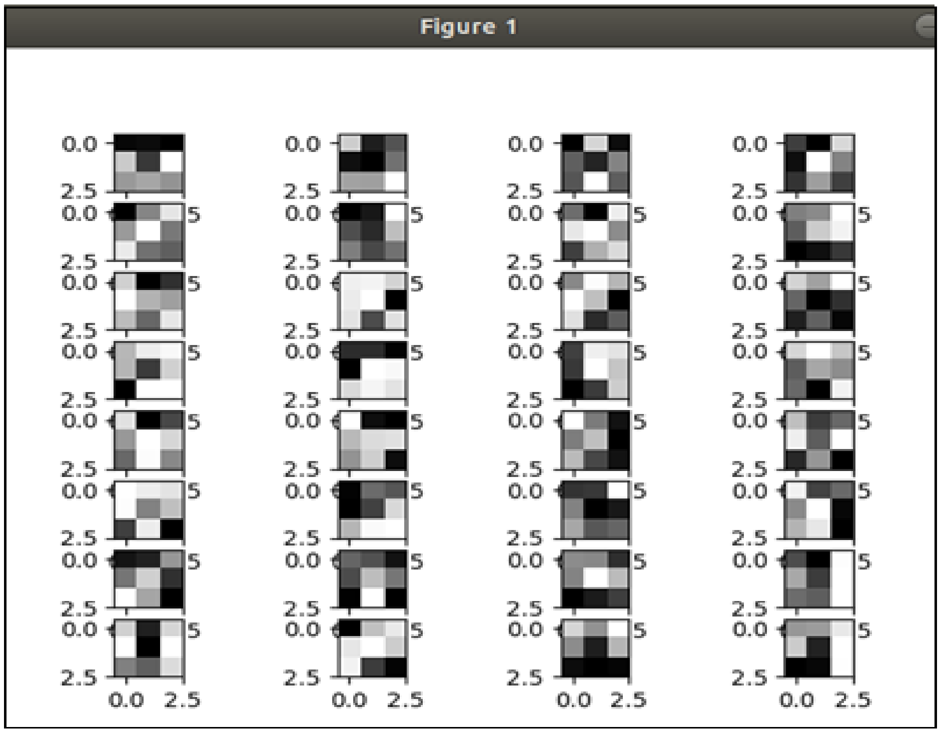


Figure 4.9: Visualization of matrix of first convolutional layer.

The above 4.9, shows a representation of filters used in the first convolutional layer of the network.

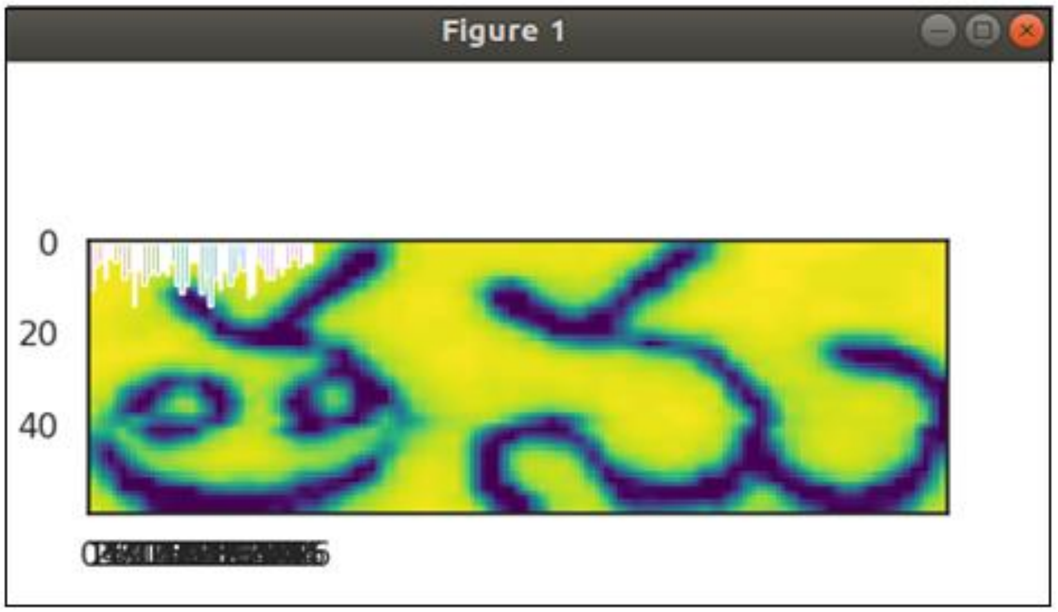


Figure 4.10: input of word image to the CNN.

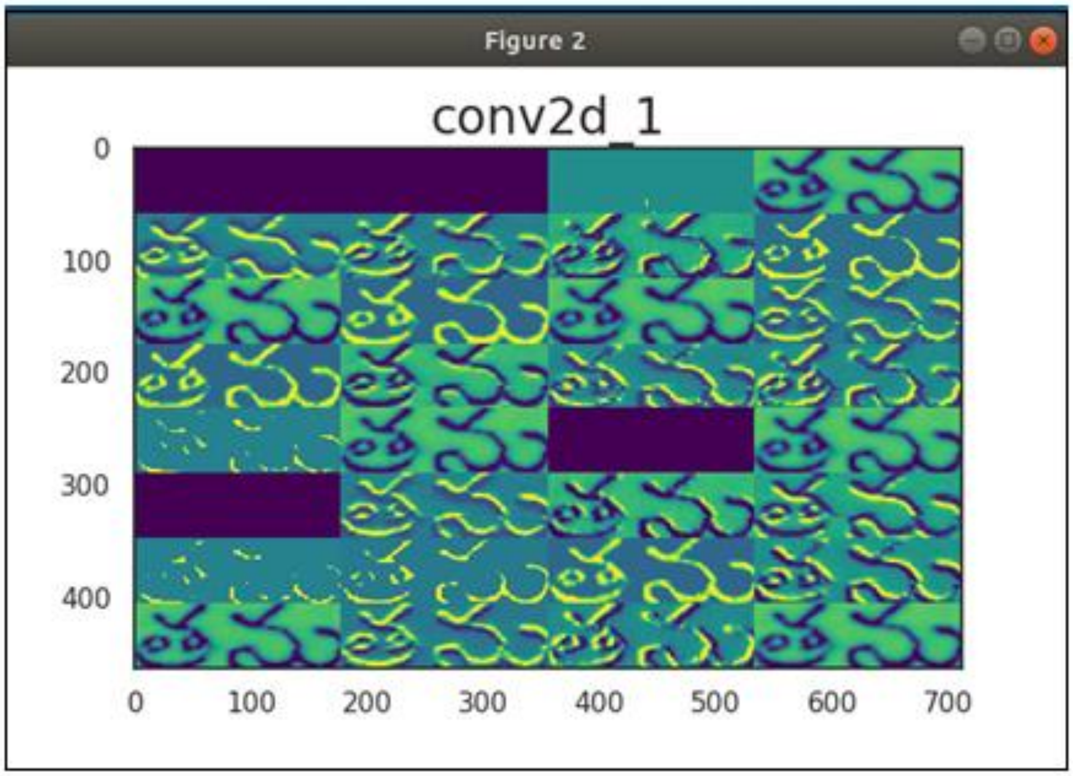
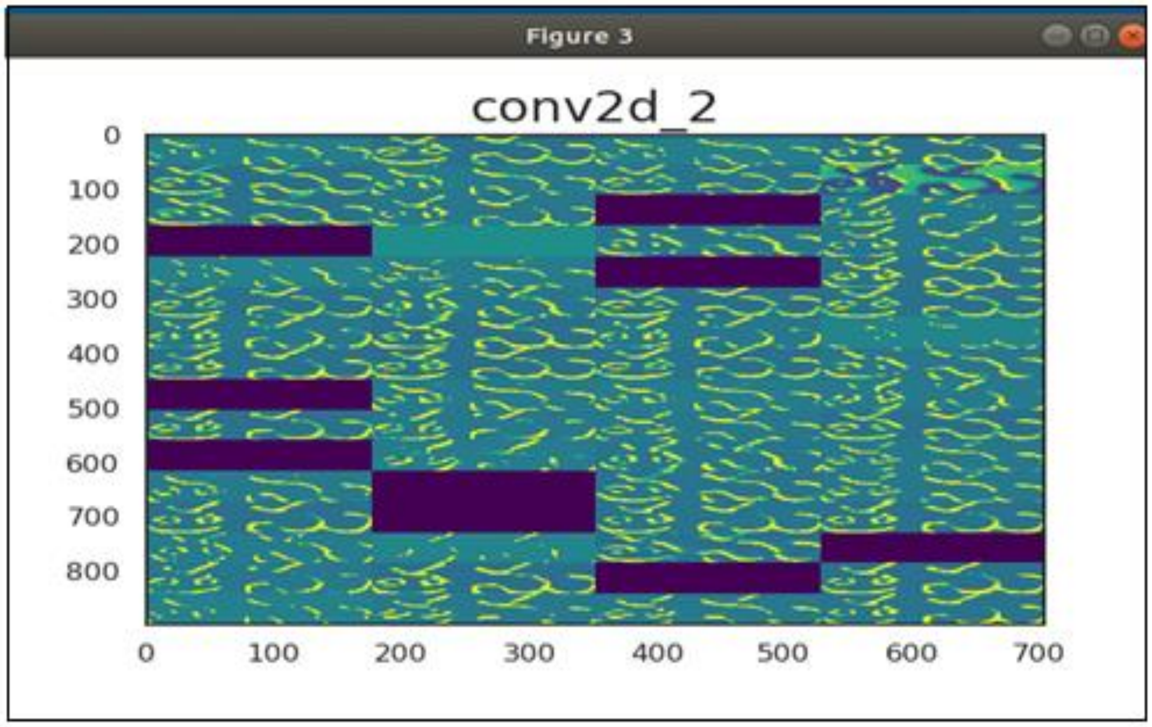
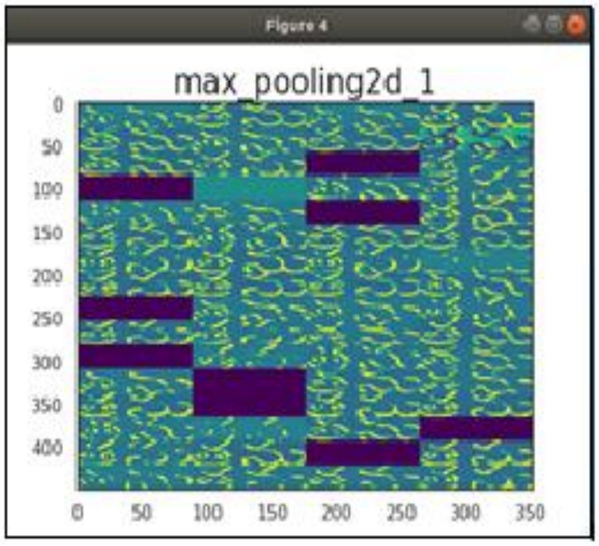


Figure 4.11: Output of first Convolutional layer.





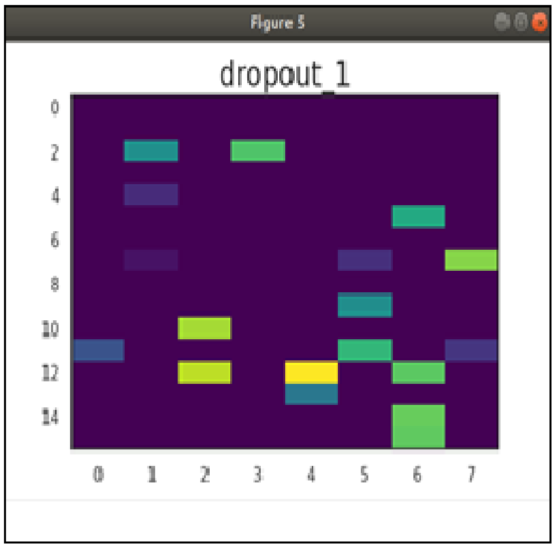


Figure 4.12: Visualization of the output of convolutional and pooling layers.

The above figure 4.12, shows the output of convolutional and pooling layers. The maxpooling layer output image with pool size of 2x2. The dropout is used to overcome the overfitting.

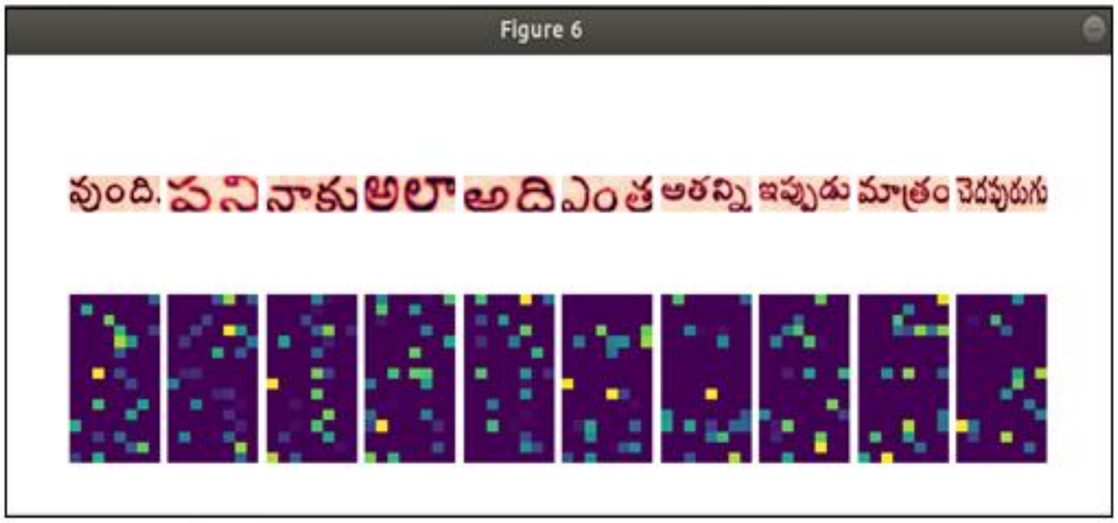


Figure 4.13: visualize the output of fully connected layer.

The above figure 4.13, shows the output of fully connected layer. Here it shows the 10 word output images with their representation.

5. CONCLUSION AND FUTURESCOPE

In this project, Design of Telugu word Recognition using CNN system is presented. Page segmentation is achieved by using run-length smearing algorithm which segregates the text and non-text regions. Segmentation involves various phases like binarization, page Deskewing, noise removal, smearing and finding the connected components and based on their features like height, the connected components are separated into words and graphics images. Telugu word Recognition is achieved by training the Convolutional neural network with 2580 training samples and 287 validation samples containing 48 labels. The training is performed in 12 epochs achieving an accuracy of 97.5% on validation data.

This project can be extended by increasing the number of training samples with more number of labels and also can be used for hand-written character recognition when trained with images of hand-written document images.

BIBLIOGRAPHY

[1] Saad ALBAWI , Tareq Abed MOHAMMED and Saad AL-ZAWI, “Understanding of a Convolutional Neural Network”, 2017.

[2] Scott Leishman, “Shape-Free Statistical Information in Optical Character Recognition”, M.S. thesis, Dept. Comp.Sci., University of Toronto, Ontario, 2007.

[3] Oivind Due Trier and Anil K. Jain, “Goal-directed evaluation of binarization method”, 1995.

[4] Viet Phuong Le, Nibal Nayef, Muriel Visani, Jean-Marc Ogier and Cao De Trant, “Text and Non-text Segmentation based on Connected Component Features”, 2015.

[5] B. Gatos and N. Papamarkos, “Applying Fast Segmentation Techniques at a Binary Image Represented by a Set of Non-Overlapping Blocks”, 2001.

[6] LeCun, Yann, “LeNet-5, convolutional neural networks”, Retrieved 2013.

[7] Ciresan, Dan; Ueli Meier; Jonathan Masci; Luca M. Gambardella; Jurgen Schmidhuber, “Flexible, High Performance Convolutional Neural Networks for Image Classification”, 2013.

[8] Matusugu, Masakazu; Katsuhiko Mori; Yusuke Mitari; Yuji Kaneda, “Subject independent facial expression recognition with robust face detection using a convolutional neural network”, 2013.

[9] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

APPENDIX

Code for segmentation of words

```
# Program to do Page Segmentation of Telugu Documents
import numpy as np
import cv2
import os
path1="/home/saikrishna/projects/majorproject/project"
path2="/home/saikrishna/projects/majorproject/outputs/connected/both"
path3="/home/saikrishna/projects/majorproject/outputs/connected/separate/text"
path4="/home/saikrishna/projects/majorproject/outputs/connected/separate/images"
#threshhold th for horizontal segmentation
th=10
for fn in os.listdir(path1):
  name=path1+'/'+fn
  img=cv2.imread(name,0)      # To read input as img

           # implement RLSA Algorithm here.
   imgarr=np.array(img)
  print(imgarr[5])
   #inversion and binarisation
  for i in range(0,len(imgarr)):
           for j in range(0,len(imgarr[i])):
                    if imgarr[i][j]<=127:
                             imgarr[i][j]=255
                    else:
                             imgarr[i][j]=0
  print("after inversion")
  print(imgarr[5])
  height,width=img.shape[ :2]
  print("height1 is"+str(height))
  print("height2 is"+str(width))
  i=0
  j=0
  # horizontal smearing
  while (i<height):
           print("hi1")
           j=0
           count=0
           revcount=0
           while j<width:
                    #print("hi2")

                    if(imgarr[i][j]==0):
                             count+=1
```

```
            j+=1
            print("i is "+str(i)+"j is "+str(j)+"count is "+str(count))
        else:
            revcount=0
            print("value at ij is"+str(imgarr[i][j]))
            if(count<=th and count!=0):
                print("before(j+count)"+str(j))
                while(revcount<count):
                    j-=1
                    imgarr[i][j]=255
                    revcount+=1
                j=j+count
            count=0
            print("after(j+count)"+str(j))
            j+=1


    i=i+1
#vertical smearing
th=20
# 20 for telugu
# 5 for english
i=0
j=0
while (i<width):
    print("hi1")
    j=0
    count=0
    revcount=0
    while j<height:
        #print("hi2")

        if(imgarr[j][i]==0):
            count+=1
            j+=1
            print("i is "+str(i)+"j is "+str(j)+"cou nt is "+str(count))
        else:
            revcount=0
            print("value at ij is"+str(imgarr[j][i]))
            if(count<=th and count!=0):
                print("before(j+count)"+str(j))
                while(revcount<count):
                    j-=1
                    imgarr[j][i]=255
                    revcount+=1
                j=j+count
```

```
                        count=0
                        print("after(j+count)"+str(j))
                        j+=1


        i=i+1

  #Connected component analysis
  output = cv2.connectedComponentsWithStats(imgarr,8,cv2.CV_32S)

print(type(output[0]))
print(output[0])
print(type(output[1]))
print(output[1])
print(type(output[2]))
print(output[2])
stats = output[2]
#print(output[2][53])
print(type(output[3]))
print(output[3])

labels=output[1]







print(stats[0][3])
print(stats[0][1])
print(stats[0][2])

#writing separate components in separate folders

for i in range(0,output[0]):
  x_cor=stats[i][0]
  y_cor=stats[i][1]
  height=y_cor+stats[i][3]
  width=x_cor+stats[i][2]
  img10 = img[y_cor:height,x_cor:width]
  ##to remove noise taking 500as area
  area=stats[i][4]
  if(i==0):
        continue

  if(area>500):
```

```
            if(stats[i][3]<80):
                    #print(img10)
                    print("hi")
                    name3 = path3+'/'+str(i)+'.tiff'
            cv2.imwrite(name3,img10)
            else:
                    print("hi")
                    name3 = path4+'/'+str(i)+'.tiff'
                    cv2.imwrite(name3,img10)
```

Code for fitting the model

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.models import load_model
import pickle
import numpy as np
import time
import matplotlib.pyplot as plt
start_time = time.time()
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

batch_size = 50     #16    #128
num_classes = 47    #10
epochs =12  #12
#d is a dictionary for labels
d={'0':0,'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9,'10':10,'11':11,'12':12,'13':13,'14':14,'15':
15,'16':16,'17':17,'18':18,'19':19,'20':20,'21':21,'22':22,'23':23,'24':24,'25':25,'26':26,'27':27,'2
8':28,'29':29,'30':30,'31':31,'32':32,'33':33,'34':34,'35':35,'36':36,'37':37,'38':38,'39':39,'40':40
,'41':41,'42':42,'43':43,'44':44,'45':45,'46':46}

# input image dimensions
  img_rows, img_cols = 60,180

  #training data
  file = open('./data.pkl','rb')
  x_train,y_train = pickle.load(file)
  file.close()
  T1=len(y_train)
  for i in range(0,T1):
          y_train[i]=d[y_train[i]]

  # Splitting training data set into "train and test = 90% and 10% respectively"

  random_seed = 2
  X_train, X_test, Y_train, Y_test = train_test_split(x_train, y_train, test_size = 0.1,
  random_state=random_seed)

  # Splitting 90% training data set into "train and validation = 80% and 10% respectively"

  X_train, X_val, Y_train, Y_val= train_test_split(X_train, Y_train, test_size=0.1,
  random_state=random_seed)

  # Mapping to "list" into "array"

  X_train = np.asarray(X_train)
  Y_train = np.asarray(Y_train)

  X_val = np.asarray(X_val)
  Y_val = np.asarray(Y_val)
  if K.image_data_format() == 'channels_first':
     X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
     X_val = X_val.reshape(X_val.shape[0], 1, img_rows, img_cols)
     input_shape = (1, img_rows, img_cols)
  else:
     X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
     X_val = X_val.reshape(X_val.shape[0], img_rows, img_cols, 1)
     input_shape = (img_rows, img_cols, 1)


  X_train = X_train.astype('float32')
  X_val = X_val.astype('float32')
  X_train /= 255
  X_val /= 255
  print('X_train shape:', X_train.shape)
  print(Y_train.shape)
  print(X_train.shape[0], 'train samples')
  print(X_val.shape[0], 'test samples')


  # convert class vectors to binary class matrices
  Y_train = keras.utils.to_categorical(Y_train, num_classes)
  Y_val = keras.utils.to_categorical(Y_val, num_classes)

  # Saving test data into a Separate file " test_data.pkl"
  file = open('./test_data.pkl','wb')
```

```
pickle.dump((X_test,Y_test),file)
file.close()

#creating the model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu', input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history=model.fit(X_train, Y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,validation_data=(X_val, Y_val))
score = model.evaluate(X_val, Y_val, verbose=0)

#generating the plots
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['val_loss'], color='r', label="validation loss",axes =ax[0])
legend = ax[0].legend(loc='best', shadow=True)
ax[1].plot(history.history['val_acc'], color='r',label="Validation accuracy")
legend = ax[1].legend(loc='best', shadow=True)

model.summary()
# Here, it is the command to save the model, can loaded later for testing.
model.save('my_model.h5')

#model = load_model('my_model.h5')

print('Validation loss:', score[0])
print('Validation accuracy:', score[1])
print("--- %s seconds ---" % (time.time() - start_time))
print("--- strart time is %s seconds ---" % (start_time))
print("--- end time is %s seconds ---" % (time.time() ))
```

Code for Visualizing the matrix of first convolutional layer

```
''' Visualize the matrix of first convolutional layer'''

import keras
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras.optimizers import Adam
from keras import backend as K
from keras.models import load_model
import pickle

from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import LearningRateScheduler
from keras import models
import pandas as pd
import numpy as np
import itertools

# plotting figures

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns

model = load_model('my_model.h5')
layer1 = model.layers[0]
layer1.name
conv2d_1w = layer1.get_weights()[0][:,:,0,:]
for i in range(1,33):
    plt.subplot(8,4,i)
    plt.imshow(conv2d_1w[:,:,i-1],interpolation="nearest",cmap="gray")
plt.show()

print("-----------------")
print("the values of the filters are ")
for i in range(1,33):
print(conv2d_1w[:,:,i-1])
```

Code for Visualizing the output of convolutional and pooling layers

```
import keras
#from keras.utils import to_categorical
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.models import load_model

import pandas as pd
import pickle
import numpy as np

import time
start_time = time.time()
# modeling
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D,
BatchNormalization
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import LearningRateScheduler
from keras import models
# plotting figures

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns

sns.set(style='white', context='notebook', palette='deep')


model = load_model('my_model12.h5')
num_classes = 47
d={'0':0,'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9,'10':10,'11':11,'12':12,'13':13,'14':14,'15
':15,'16':16,'17':17,'18':18,'19':19,'20':20,'21':21,'22':22,'23':23,'24':24,'25':25,'26':26,'27':2
7,'28':28,'29':29,'30':30,'31':31,'32':32,'33':33,'34':34,'35':35,'36':36,'37':37,'38':38,'39':39,'
40':40,'41':41,'42':42,'43':43,'44':44,'45':45,'46':46}

img_rows, img_cols = 60,180

file = open('./test_data.pkl','rb')
```

```
X_test,Y_test = pickle.load(file)
file.close()


T1=len(Y_test)

#2 graph representation

g = sns.countplot(Y_test)

Y_test_value=Y_test  ## keep the original label


# Mapping to "list" into "array"

X_test = np.asarray(X_test)
Y_test = np.asarray(Y_test)

if K.image_data_format() == 'channels_first':
   X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
   input_shape = (1, img_rows, img_cols)
else:
   X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
   input_shape = (img_rows, img_cols, 1)

X_test = X_test.astype('float32')

X_test /= 255

print('X_test shape:', X_test.shape)
print(Y_test.shape)
print(X_test.shape[0], 'test samples')

Y_test = keras.utils.to_categorical(Y_test, num_classes)

test_im = X_test[2]
plt.imshow(test_im.reshape(img_rows, img_cols), cmap='viridis', interpolation='none')

# activation output
layer_outputs = [layer.output for layer in model.layers[:8]]
activation_model = models.Model(input=model.input, output=layer_outputs)
activations = activation_model.predict(test_im.reshape(1,img_rows, img_cols,1))

layer_names = []
print("layers are")
for layer in model.layers:
    layer_names.append(layer.name)
    print(layer.name)
print(len(layer_names))
images_per_row = 4

#code of 5.2 is on commenets below that 5.3 is present

for layer_name, layer_activation in zip(layer_names[:4], activations):
    #if layer_name.startswith('conv'):
print(layer_activation)
        n_features = layer_activation.shape[-1]
print(n_features)
        size = layer_activation.shape[1]

size1 = layer_activation.shape[2]
#print(layer_activation.shape[2])
print(layer_activation.shape)
print("line105")
#size1 = layer_activation.shape[2]
print("line107")

            print(layer_activation.shape[0])
            print("size is "+str(size))

 n_cols = n_features // images_per_row
   display_grid = np.zeros((size * n_cols, images_per_row * size1))  ## * size
            print(len(display_grid))
for col in range(n_cols):
    for row in range(images_per_row):
          channel_image = layer_activation[0,:, :, col * images_per_row + row]
            print("len of channle image is")
            print (len(channel_image))
            l = len(channel_image)
            len1= len(channel_image[0])
            for i in range(0,l):
                        print(len(channel_image[i]))

            channel_image -= channel_image.mean()
            #print(channel_image)
             channel_image /= channel_image.std()
            #print(channel_image)
             channel_image *=64
            #print(channel_image)
             channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype('uint8')
             display_grid[col * size : (col + 1) * size,
                row * size1 :(row + 1) * size1] = channel_image
 scale = 1 / size
        plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
        plt.title(layer_name, fontsize = 20)
        plt.grid(False)
 plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

Code for Visualizing the output of fully connected layer

```
fc_layer = model.layers[-3]
activation_model = models.Model(input=model.input, output=fc_layer.output)
activations = activation_model.predict(test_im.reshape(1,60,180,1))
print("activations are")

print(len(activations[0]))
print(activations[0])
activation = activations[0].reshape(16,8)    ### activations[0]len is 128 so toook   16,8
                                             #if model.layers[-4] is take n then 157696=448,352
plt.imshow(activation, aspect='auto', cmap='viridis')

# organize the testing images by label
Y_test_value_df = pd.DataFrame(Y_test_value,columns=['label'])
Y_test_value_df['pos']=Y_test_value_df.index
Y_test_label_pos = Y_test_value_df.groupby('label')['pos'].apply(list)
pos = Y_test_label_pos[1][0]

#display 3 rows of digit image [0,9], with last full connected layer at bottom
plt.figure(figsize=(16,8))
x, y = 10, 1
for i in range(y):
    for j in range(x):
        # word image
        plt.subplot(y*2, x, i*2*x+j+1)
        pos = Y_test_label_pos[j][i] # j is label, i in the index of the list
        plt.imshow(X_test[pos].reshape((img_rows, img_cols)),interpolation='nearest')
        plt.axis('off')
        plt.subplot(y*2, x, (i*2+1)*x+j+1)
   activations = activation_model.predict(X_test[pos].reshape(1,img_rows, img_cols,1))
   activation = activations[0].reshape(16,8)
  plt.imshow(activation, aspect='auto', cmap='viridis')
  plt.axis('off')

plt.subplots_adjust(wspace=0.1, hspace=0.1)
plt.show()
```