ABSTRACT

Hand Writing Recognition refers to recognizing a handwritten letter and predicts the letter correctly.There exist many applications for handwriting recognition.Some are Reading postal addresses,Automatic conversion of text in an image into letter codes,Recognizing a Hand written letter in one language and predicting its equivalent letter in another language etc.Many surveys on Hand writing Recognition results conversion of hand written text to machine readable form.For recognizing a letter,Machine Learning is enough.Coming to recognition of words,Deep Learning comes into picture.There exists some algorithms to divide words into letters and predicts them correctly.In recent days,there is an exponential growth in the numbers of researches carried out in this field because of its benefits.On the past decade,great efforts are paid to the online hand written content interpretation from hand written text.In this document, a detail of Hand Written Recognition is carried out including the recent works and advancements.Here this document consists of several modules:Feature,Character Extraction and its Description, methods,Language Models and also have a detail implementation of Telugu Hand-Written Letter Recognition and its Equivalent English Letter Prediction.

TABLE OF CONTENTS:

- Result

- **Recent Advancements: Introducing CapsNets**
  - Beyond text to text
    - Text to Speech
    - Text to Images
- **Implementation of Telugu Hand-Written Letter Recognition and its Equivalent English Letter Prediction using Machine Learning**
- **Conclusions**

INTRODUCTION

Handwriting Recognition is a challenging issue because of the large data variability.All the modern inventions in computer and communication technologies such as word processors, fax machines and e-mail are having their impact on handwriting. These in-variations have led to the fine-tuning and reinterpreting of the role of handwriting and handwritten messages.It is an open research issue to recognize the handwritten words out of scanned documents which is unconstrained. Document complex structures hinder sections into words and lines. For recognizing word, the handwriting variability implicates other challenging level additionally.Several types of analysis, recognition, and interpretation can be associated with handwriting. Handwriting recognition is the task of transforming a language re-presented in its own spatial form of graphical marks into a symbolic representation.” A common complaint and excuse of people is that they couldn’t read their own handwriting. So what chance does a computer have?”

Handwriting data is converted to digital form either by scanning the writing on paper or by writing with a special pen on an electronic surface. The two approaches are distinguished as offline and on-line handwriting, respectively. In the on-line case, the two-dimensional co- ordinates of successive points of the writing as a function of time are stored in order. In the off-line case, only the completed writing is available as an image. Figure 1(a) and 1(b) shows the analysis of the two cases. The recognition rates reported are much higher for the on-line case in comparison with the off-line case. Off-line systems are less accurate than on-line systems. However, they are now good enough that they have a significant economic impact for specialized domains such as interpreting hand-written postal addresses on envelopes and reading courtesy amounts on bank checks.

Fig. 1(a)

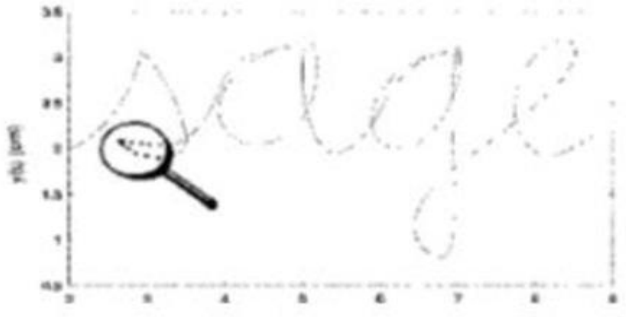


Fig. 1(b)

On robotic recognition of characters, Lexicon independent techniques and Segmentation free/Segmentation based model are used by employing its global features, segmentation free methods try and distinguish the whole word image. At the same time, segmentation-based methods, depend on word breaking image into smaller sections recognizable as characters and link a character label to these sections. By employing a time series models like Hidden Markov Models, contextual dependencies among nearest segments are used.In contrast to this Conditional Random Fields model, the conditional distribution does not

create any considerations over the data distribution. Conditional Random Fields can obtain sum of feature functions unlike Hidden Markov Models and every feature function can employ the whole sequence of input data. A basic restriction of maximum entropy Markov models can be avoided by Conditional Random Fields and other discriminative Markov models depending on directed graphical models that can be influenced to states with some successor states. The model Hidden Markov Models obtains the transition to the similar label of the character otherwise to the subsequent word character and it can be employed for recognizing free word segments.

Offline Handwriting Recognition:

The central tasks of off-line handwriting recognition are character recognition and word recognition. Document analysis is the necessary preliminary step in recognition that locates appropriate text when complex, two-dimensional spatial lay-outs are employed. Different approaches have been proposed to offline recognition that have contributed to the present day efficiency of the technique.

Preprocessing:

It is necessary to perform several document analysis operations prior to recognizing text in scanned documents. Some of the common operations performed prior to recognition are: thresholding, the task of converting a gray-scale image into a binary black-white image; noise removal, the ex-traction of the foreground textual matter by removing, say, textured background, salt and pepper noise and interfering strokes; line segmentation, the separation of

individual lines of text; word segmentation, the isolation of textual words, and character segmentation, the isolation of individual character, typically those that are written discretely rather than cursively.

Thresholding:

The task of thresholding is to extract the foreground (ink) from the background (paper). The histogram of gray-scale values of a document image typically consists of two peaks: a high peak corresponding to the white background and a smaller peak corresponding to the fore- ground. So, the task of determining the threshold gray-scale value is one of determining an "optimal" value in the valley between the two peaks. The distributions of the foreground and background points are regarded as two classes. Each value of the threshold is tried and one that maximizes the criterion is chosen. There are several improvements to this basic idea, such as handling textured backgrounds similar to those encountered on bank checks.

Noise Removal:

Noise removal is a topic in document analysis that has been dealt with extensively for typed or machine-printed documents. For handwritten documents, the connectivity of strokes has to be preserved. Digital capture of images can introduce noise from scanning devices and transmission media. Smoothing operations are often used to eliminate the articrafts introduced during image capture. One study, describes a method that performs selective and adaptive stroke "filling" with a neighborhood operator which emphasizes stroke connectivity, while at the same time, conservatively check aggressive "over-filling."

Line Segmentation:

Segmentation of handwritten text into lines, words, and characters has many sophisticated approaches. This is in contrast to the task of segmenting lines of text into words and characters, which is straight-forward for machine-printed documents. It can be accomplished by examining the horizontal histogram profile at a small range of skew angles. The task is more difficult in the handwritten domain. Here, lines of text might be undulate up and down and ascenders and descenders frequently intersect characters of neigh-boring lines. One method is based on the notion that people write on an imaginary line which forms the core upon which each word of the line resides. The local minima points approximate this imaginary baseline from each component. A clustering technique is used to group the minima of all the components to identify the different handwritten lines.

Word and Character Segmentation:

Line separation is usually followed by a procedure that separates the line into words. Few approaches in the literature have dealt with word segmentation issues. Among the ones that have dealt with segmentation issues, most focus on identifying physical gaps using only the components. These methods assume that gaps between words are larger than the gaps between characters. However, in hand-writing, exceptions are commonplace be-cause of flourishes in writing styles with leading and trailing ligatures. Another method incorporates cues that humans use and does not rely solely on the one-dimensional distance between components. The author’s writing styles, in terms of spacing, is captured by characterizing the variation of spacing between adjacent characters as a

function of the corresponding characters themselves. The notion of expecting greater space between characters with leading and trailing ligatures is enclosed into the segmentation scheme.

Before Deep Learning, there were OCRs

Handwritten text classifiers were first required for classification of postal mail. Using scanning equipment, hardwired logic recognised mono-spaced fonts. The first Optical Character Recognition (OCR) software developed in 1974 by Ray Kurzweil. By reducing the problem domain, the process was more accurate. This allowed for recognition in handwritten forms. Foremost, it lacked efficiency and knowledge of unexpected characters. These classical techniques carried heavy limitations in two key areas:

Character extraction — Individual characters are recognised by ease with OCR. Cursive handwriting, which is connected, poses more issues with evaluation. It is difficult to interpret handwriting with no distinct separation between characters.

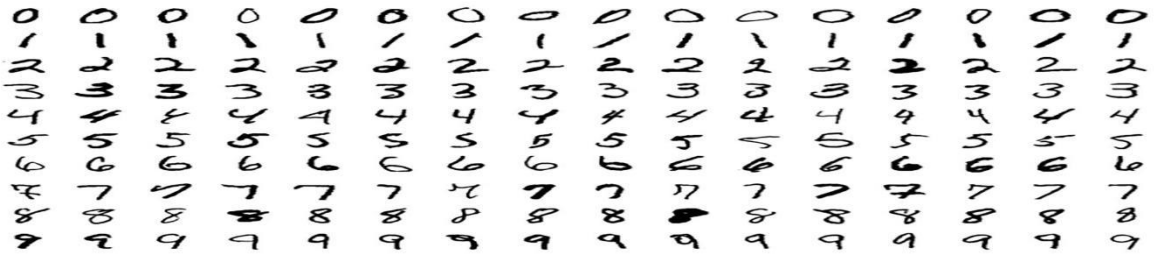
Feature extraction — Individual properties of symbols were hard-coded, and matched to input symbols. Properties include aspect ratio, pixel distribution, number of strokes, distance from the image centre, and reflection. This requires development time, as these properties are added manually.

Using Neural Networks

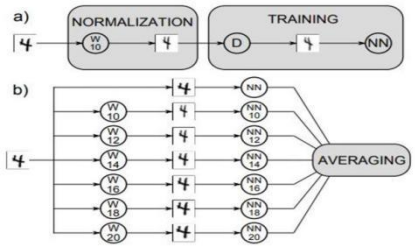
Neural networks are able to learn features from analysing a dataset, and then classify an unseen image based on weights.Features are extracted in the convolutional layers, where a kernel is passed over the image to extract a certain feature. In the end result, multiple kernels learn all the features within a dataset, in order to make classifications. This solves the issue of feature extraction in OCR methods.

Furthermore, using neural networks on a database, rather than classical methods, means no manual hardcoding is needed. Instead, parameters are learnt during the training process. This makes deep learning methods more resilient to changes in handwriting styles, and alleviates the challenges in feature extraction in classical methods. However, the output accuracy depends strongly on the quality and completeness of the dataset used in the training process.

The creation of the MNIST Database propelled research towards using neural networks for handwriting recognition. The database contains 70,000 handwritten digits, and has been used in deep learning since 1998. In LeCuns' pioneering paper, heads were turned by the introduction of neural networks for handwriting. Using LeNet-5, and distorting the MNIST digits, an error rate of 0.7% was immediately achieved — A vast improvement from classical methods.

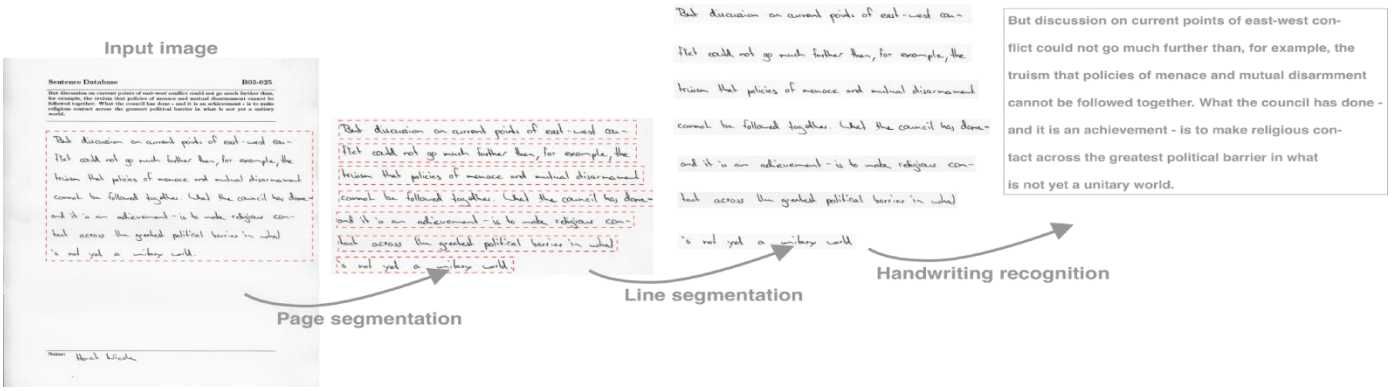


Beyond this, neural networks have been used to classify even unseen alphabets. This means, models can be generalised for any language, and does not require training on a specific character database, such as MNIST. Graves produced 91.4% accuracy on an unseen arabic characters.The use of convolutional neural networks (CNNs) peaked in 2011, when Ciresan analysed handwriting, achieving a tiny 0.27% error rate. To do this, seven deep CNNs trained identical classifiers on data, pre-processed in different ways. This was made more achievable by advances in hardware, where powerful GPUs handle deep learning tasks efficiently, and are now widely used in the community. The errors differed as much as possible, and outputs were averaged. The results are comparable to human-like performance.



Character Recognition:

In Machine Learning, Recognizing the Character is quiet different from Deep Learning.Its easy to predict a character rather than a word in Machine Learning.A pattern recognition algorithm is used to extract shape features and to assign the observed character to the appropriate class. Artificial neural networks have emerged as fast methods for implementing classifiers for Optical Character Recognition. Recognition of a character from a single, machine-printed font family on a well printed paper document can be done very accurately. Difficulties arise when handwritten characters are to be handled. In difficult cases, it becomes necessary to use models to constrain the choices at the character and word levels. Such models are essential in handwriting recognition due to the wide variability of hand printing and cursive script.



Given a handwriting sample, a set of characters is first segmented, then for each isolated character, the so-called micro-features are extracted. Therefore, each handwriting sample is characterized by a number of micro-feature vectors corresponding to the characters available from the sample. Micro-features have been successfully used for recognizing handwritten characters and analyzing handwriting individuality.

Word Recognition:

A word recognition algorithm attempts to associate the word image to choices in a lexicon. Typically, a ranking is produced. This is done either by the analytic approach of recognizing the individual characters or by holistic approach of dealing with the entire word image. The latter approach is useful in the case of

touching printer characters and hand-writing. A high level of performance is observed by combining the results of both approaches. There exist several different approaches to word recognition using a limited vocabulary. One method of word recognition based on determining pre segmentation points followed by determining an optimal path through a state transition diagram. Applications of automatic reading of postal addresses, bank checks, and various forms have triggered a rapid development in handwritten word recognition in recent years. While methods have differed in the specific utilization of the constraints provided by application domain, their underlying core structure is the same. Typically, the methodology involves processing, a possible segmentation phase which could be avoided if global word features are used, recognition and post-processing. The upper and lower profiles of word image are represented as a series of vectors describing the global contour of the word image and bypass the segmentation phase.

The methods of feature extraction are central to achieving high performing word recognition. One approach utilizes the idea of “regular and “singular” features. Handwriting is regarded as having a regular flow modified by occasional singular embellishments. A common approach is to use an HMM to structure the entire recognition process. Another method deals with a limited size dynamic lexicon. Words that are relevant during the recognition task are not available during training because they belong to an unknown subset of a very large lexicon. Word images are over segmented such that after the segmentation process no adjacent characters remain touching. Instead of passing on combi- nations of segments to a generic OCR, a lexicon is brought into play early in the process. A combination of adjacent segments is compare to only those character choices which are possible at the position in the word

being considered. The approach can be viewed as a process of accounting for all the segments generated by a given lexicon entry. Lexicon entries are ordered according to the "goodness" of the match. Dynamic Programming (DP) is a commonly used paradigm to string the potential character candidates into word candidates; some methods combine heuristics with DP to disqualify certain groups of primitive segments from being evaluated if they are too complex to represent a single character. The DP paradigm also takes into account compatibility between consecutive character candidates.

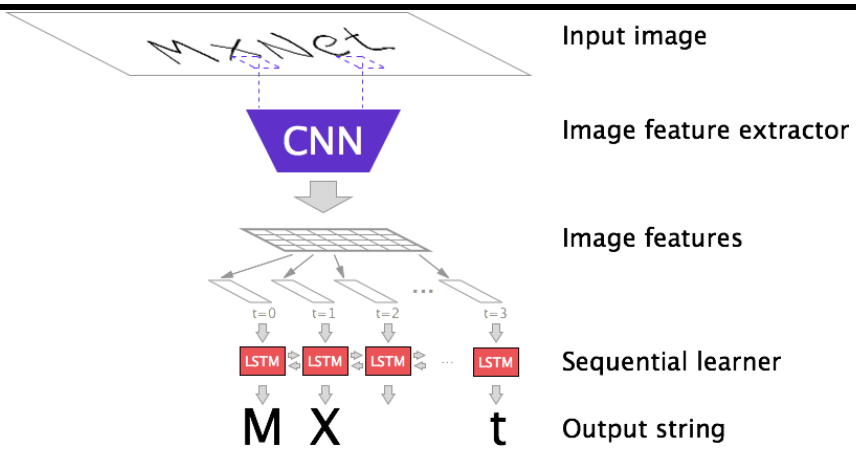
Methods:

The pipeline of this component is presented in fig-5. The input is an image containing a line of text and the output of the module is a string of the text.

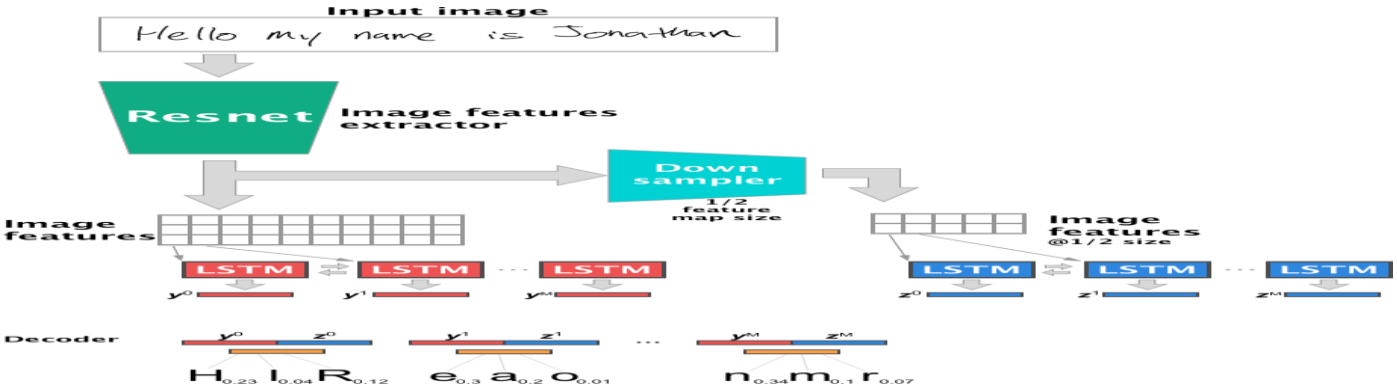
Handwriting detection:

The handwriting detection takes the input image containing a line of text and returns a matrix containing probability of each character appearing. Specifically, the matrix is of size (sequence_length × vocab. of characters).

Previous works utilize multidimensional LSTMs to recognize handwriting however more recent works suggest that similar performances can be achieved with a 1D-LSTM. Briefly,utilized a CNN for image feature extraction and fed the features into a bidirectional LSTM and trained the network to optimize the Connectionist Temporal Classification (CTC) loss (shown in Figure). Intuitively, the CNN generates image features that are spatially aligned to the input image. The image features are then sliced along the direction of the text and sequentially fed into an LSTM. This network is denoted as CNN-biLSTM. The main reason that the CNN-biLSTM was selected is because multidimensional LSTMs substantially more computationally expensive compared to the CNN-biLSTM.



I extended the described implementation by providing multiple downsamples of the image features (Figure 4).Multiple downsamples were provided to assist in recognizing images of handwritten text that vary in size (e.g., lines that contain only 1 word vs lines that contain 7 words). Note that a pre-trained res-net was used as a image feature extractor.



The output of the CNN-biLSTM is fed into a decoder to predict probability distribution over the characters for each vertical slice of the image. In the next section, methods to extract the most probably sentence from the matrix using a language model are discussed.

Language models

We explored three methods to extract a string of words given a matrix of probabilities are explored: 1) greedy search, 2) lexicon searching, and 3) beam searching + lexicon searching with a simple language model.

Greedy search

The greedy search method simply iterates over each time-step and obtains the most probable character at each time step.This method doesn't use any external language models or dictionary to alter the results.

Lexicon searching

The lexicon search model use the output of the greedy method and attempts to match each word with a dictionary. Specifically, the following procedures were taken: 1) Decontraction of the string, 2) Tokenization of the string, 3) for each word: 3.1) Check if the proposed word is an English word, if not, suggest possible words, and 3.2) Choose the word with the shortest weighted edit distance, 4) Contract texts back to its original form.

1. Decontraction of the string

Contractions are a shorted version of a word or a spoken form of a word. For example, did not → didn't, might have → might've. In order to account for contractions, pycontractions was used. This library provides contractions and decontractions of the string. For decontracting ambiguous cases (e.g., ain't → am not/are not), the library utilizes a language model (google news) to assess the correct case.

2. **Tokenization of the string**

Tokenization separates a string containing a sentence into distinct words. The tokenize package from the nltk toolkit was used to perform this. This was followed by iterating through each word.

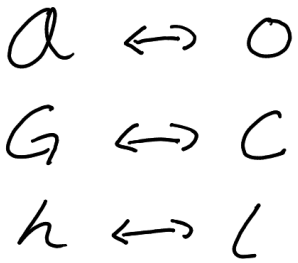
3.1 Suggest words if it's not a word

For each word that was tokenized, the pyenchant module is used to check if the word is an actual English word. If the word is an actual word, then no changes

will be made, otherwise, the norvig spell checker was used to provide suggestions for possible words.

3.2 Select the suggested word with the shortest weighted edit distance

Given a list of suggested words, the weighted edited distance between the given word and the suggested words were calculated. Specifically, the edit distance was weighted on the differences between the handwritten characters.



Characters that are shown in Figure 5 should have lower weights compared to characters that are visual different (e.g., "*w*" and "*o*") and the visually different characters should have large weights.

The character similarity was modeled using the output of the CNN-biLSTM given the training examples. The frequency of differences (insertions, deletions, and substitutions) between actual words and the predicted words was counted. The larger the frequency of a mistake, the more visually similar characters are. Therefore, the smaller the weight should be. The main issue is that the weighted edit distance is limited to substitutions of one character to another. In handwriting recognition, it is common for compounds to characters to look alike (e.g., "*rn*" and "*m*").

3. **Contract texts back to its original form**

This step simply detokenizes and contracts the words (if it was originally decontracted).

As a whole, the lexicon spell checker works best for predicted strings that are very close to the actual value. However, if the output of the greedy algorithm doesn't provide close enough string proposals, the lexicon spell checker can

actually reduce accuracies instead of improving it. A final check to ensure that the input and output sentences are similar was performed.

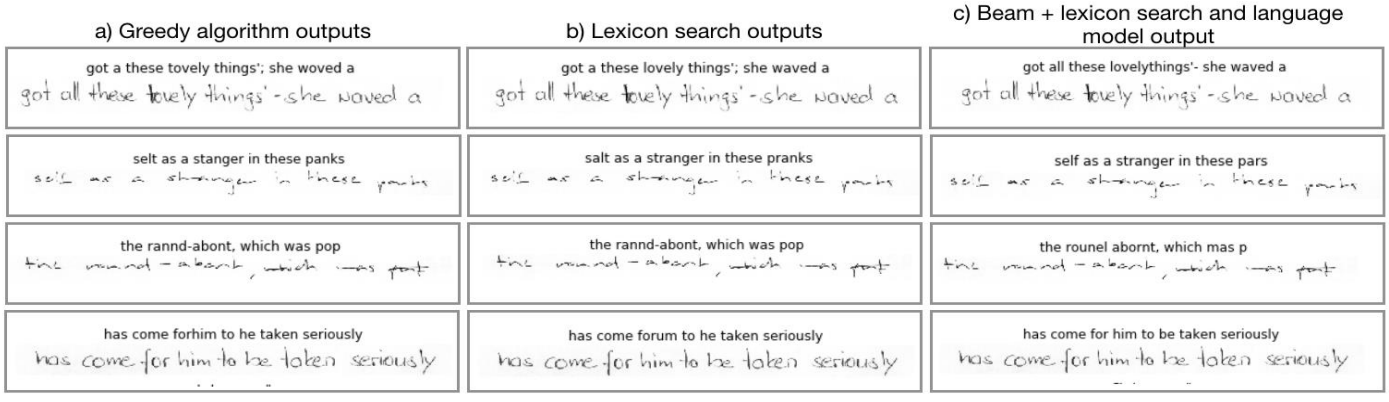
Beam searching + lexicon search + language model

In order to alleviate the issues of getting poor proposals from the greedy algorithm, one can iterate through the probability matrix to obtain multiple proposals. However, ranking all possible proposals is computationally expensive. Graves et al. [5] proposed that the beam search algorithm can be used to generate *K* proposals of strings given the probability matrix.

n our implementation, the *K* sentences are then fed into the lexicon spell checker to ensure that words can be found in a dictionary (otherwise it will appear as <unk>). Each *K* proposal is then tokenized and sequentially fed into a language model to evaluate the perplexity of each sentence proposal. The sentence proposal with the lowest perplexity was chosen.

Result:

The algorithm was qualitatively and quantitatively evaluated. Examples of the predicted text from four forms are shown in Figure 6.



The greedy, lexicon search, and beam search outputs present similar and reasonable predictions for the selected examples. In Figure 6, interesting examples are presented. The first line of Figure 6 show cases where the lexicon search algorithm provided fixes that corrected the words. In the top

example, *"tovely"* (as it was written) was corrected *"lovely"* and *"woved"* was corrected to *"waved".* In addition, the beam search output corrected "*a*" into "*all*", however it missed a space between "*lovely*" and "*things*". In the second example, "*selt*" was converted to "*salt*" with the lexicon search output. However, "*selt*" was erroneously converted to "*self*" in the beam search output. Therefore, in this example, beam search performed worse. In the third example, none of the three methods significantly provided comprehensible results. Finally, in the forth example, the lexicon search algorithm incorrectly converted "*forhim*" into "*forum*", however the beam search algorithm correctly identified "*for him*".

Quantitatively, the greedy algorithm had a mean character error rate (CER) = 18.936 whereas the lexicon search had CER = 18.897. Without the weighted edit distance, the lexicon search had CER = 19.204, substantially reducing the performance of algorithm. As presented in Figure 6, the CER improvement between using the greedy algorithm and the lexicon search was minimal. As expected, the beam search algorithm out performed the greedy and the lexicon search results with CER = 18.840.

Introducing CapsNets

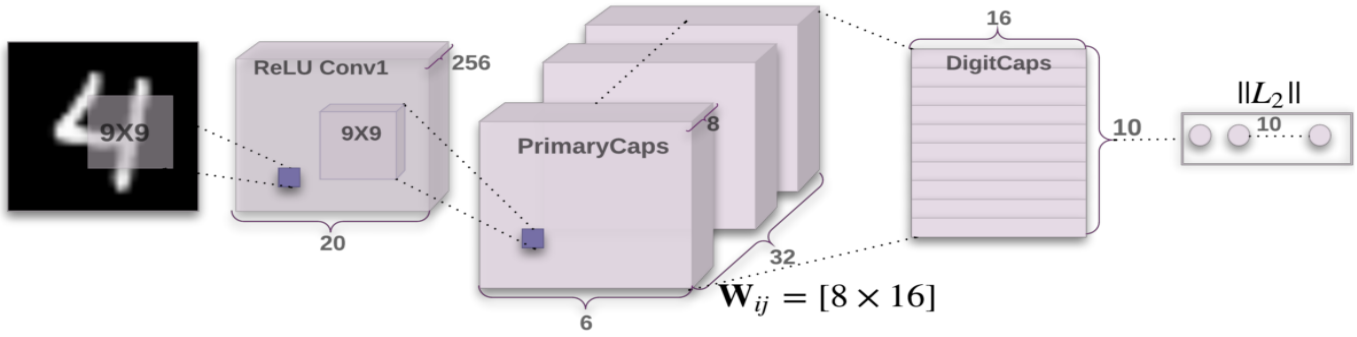
Emerging research obtained into Capsule Networks (CapsNets) has led to further advancements in the field. As another approach to machine learning solutions, the introduction of CapsNets shows a future of improvements for handwriting recognition. Although fairly new on the scene, applications of CapsNets are beginning to pick up the pace, and absolve some of the limitations of CNNs.

CapsNets improve on the results of CNNs in a number of ways, particularly the following:

1. **Reduces the effects of spatial variance**, found using CNN methods.
2. **Reduces the amount of data** required for training.

In CNNs, the spatial variance of different kernels has been a prominent issue. To resolve this using CapsNets, kernels used to determine features work together, with the aid of dynamic routing, to combine individual opinions of multiple

groups (capsules). This results in equivariance among kernels, and greatly improves performance in comparison to CNNs.



To most effectively use CNNs for the purpose of handwriting recognition, it is necessary to have large datasets containing handwritten characters. The datasets need to be large, as the model needs to learn a large amount of variance, to accommodate for different handwriting styles. CapsNets help reduce the amount of data required, whilst still maintaining high accuracy. For instance, a recent team was able to train with a mere 200 training samples per class, surpassing or achieving CNN character recognition results, whilst also able to recognise highly overlapping digits.

Beyond text to text

So far, this article has touched on generating a digital text output from a handwritten input. Now that handwritten text has been successfully digitised, further steps can be taken to generate different forms of output.

Text to speech

By coupling a text input with the spoken voice of a person, machine learning techniques has allowed for human-like text to speech. For example, Googles WaveNet introduces mel spectrograms as an input to the network, which dictates how words are pronounced, but also indicates volumes and intonations. This allows for adding emphasis on capitalised words, pauses for punctuation, whilst also compensating for spelling mistakes in the original text. The resulting speech sounds natural and human.

Text to Images

Deep learning methods can now turn natural language into synthesised images. Here, written descriptions for images are used to build a new, hallucinated image. This is done by training a GAN, to create realistic high resolution images from noise, until it convincingly matches the text input.

The result generates an image which depicts the content of the written text. The below figure shows examples, where the text input has been passed through a GAN at Stage-I, to generate an image. The resolution is then improved at Stage-II, to create the final synthesised image.

Implementation of Telugu Hand-Written Letter Recognition and its Equivalent English Letter Prediction using Machine Learning:

* Data-set is developed for prediction of (A,E,I,O,U) English Letters.it contains more than 1200 samples of 5 Telugu hand-written letters.

* Train size: 80% and Test size: 20%

* Used Support Vector Machine Algorithm

* Web Development using Flask

* Accuracy: >99%

* Input: Telugu Hand-Written Letter

* Output: Equivalent English Letter

Code:

model.py:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pickle
from sklearn.svm import SVC
from sklearn import decomposition
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import metrics
from PIL import Image
import cv2


import os


def createFileList(myDir, format='.png'):
    fileList = []
    print(myDir)
    for root, dirs, files in os.walk(myDir, topdown=False):
        for name in files:
```

```
                if name.endswith(format):
                        fullName = os.path.join(root, name)
                        fileList.append(fullName)
        return fileList



df = pd.read_csv('teluguletters.csv')


pix=[]
for i in range(1,10001):
        pix.append('pix-'+str(i))
features=pix
X = df.loc[:, features].values
y = df.loc[:,'class'].values


X_train, X_test, y_train, y_test = train_test_split( X, y, test_size = 0.2,
random_state = 100)
y_train=y_train.ravel()
y_test=y_test.ravel()
svm_model = SVC(kernel = 'linear', C = 1, gamma=0.0001).fit(X_train, y_train)
```

```
        # Saving model to disk
pickle.dump(svm_model, open('mymodel.pkl','wb'))


        # Loading model to compare the results
model = pickle.load(open('mymodel.pkl','rb'))
# load the original image
myFileList = createFileList(r'C:\Users\HP\Music\MLproject\images')
for i in myFileList:
        new_X = cv2.imread(i)
        new_X = cv2.cvtColor(new_X, cv2.COLOR_BGR2GRAY)
        img =cv2.resize(new_X, ( 100, 100), interpolation=cv2.INTER_AREA)
        img = img/255.0
        plt.imshow(img,cmap = "gray")
        img = img.ravel()
        new_y = model.predict([img])
        print("predicted label",int(new_y))
        #Another Way
        '''img_file = Image.open(i)

                # img_file.show()


        width, height = img_file.size
        format = img_file.format
```

```
        mode = img_file.mode


                    # Make image Greyscale

        img_grey = img_file.convert('L')

                   #img_grey.save('result.png')

                    #img_grey.show()


                    # Save Greyscale values

        value = np.asarray(img_grey.getdata(),
dtype=np.int).reshape((img_grey.size[1], img_grey.size[0]))/255

        value = value.flatten()

        value=value.reshape(1,-1)




        print()

        print(model.predict(value))

        print()'''

accuracy = svm_model.score(X_test, y_test)

print('Accuracy: ',accuracy*100)
```

MainApplication.py:

```
import os

from PIL import Image
```

```
# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import pickle

from sklearn.svm import SVC

from sklearn import decomposition

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

from sklearn import metrics

from PIL import Image


from flask import Flask, render_template, request, redirect, url_for,
send_from_directory

import base64, os

import numpy as np

import time

import cv2

import os, random

import sys

import os

import csv
```

```
app = Flask(__name__)

APP_ROOT = os.path.dirname(os.path.abspath(__file__))
model = pickle.load(open('mymodel.pkl','rb'))

@app.route('/')
def index():
    return render_template("Recognition.html")
@app.route('/predict',methods=['POST'])

def predict():
    return render_template("upload.html")



@app.route('/upload', methods=['POST'])

def upload():
    target = os.path.join(APP_ROOT, 'images/')
    print(target)

    if not os.path.isdir(target):
```

```
        os.mkdir(target)


    for file in request.files.getlist("file"):
        print(file)
        '''new_X = cv2.imread(file)
        new_X = cv2.cvtColor(new_X, cv2.COLOR_BGR2GRAY)
        img =cv2.resize(new_X, ( 100, 100), interpolation=cv2.INTER_AREA)
        img = img/255.0
        plt.imshow(img,cmap = "gray")
        img = img.ravel()
        new_y = model.predict(img)'''
        filename = file.filename
        destination = "/".join([target, filename])
        print(destination)
        file.save(destination)
        img_file = Image.open(file)
        width, height = img_file.size
        format = img_file.format
        mode = img_file.mode
        img_grey = img_file.convert('L')


        value = np.asarray(img_grey.getdata(),
dtype=np.int).reshape((img_grey.size[1], img_grey.size[0]))/255
```

```
            value = value.flatten()
            value=value.reshape(1,-1)
            var=int(model.predict(value))
            let=''
            if var==0:
                let='A'
            elif var==1:
                let='E'
            elif var==2:
                let='U'

            elif var==4:
                let='O'
            else:
                let='I'
            print()

        return render_template("prediction.html",prediction_text='predicted
Letter:    "{}"'.format(let))

if __name__ == "__main__":
        app.run(port=4555, debug=True)
```

Recognition.html:

```
<!DOCTYPE html>
<html >
<html lang="en">
   <head>

       <meta charset="UTF-8">
       <meta name="viewport" content="width=device-width, initial-scale=1.0">
       <meta http-equiv="X-UA-Compatible" content="ie=edge">
       <title>Recognize Letters!</title>
<style><head>
* {
   box-sizing: border-box;

}

main {
   width: 200px;
   height:100px;
   border: 5px solid #e0e0e0;
   margin: 0 auto;
   display: flex;

   flex-grow: 1;
```

```
}


.left-block {
    width: 160px;    background-color:#ffffcc;


    border-right: 5px solid black;
}
.crosshair              { cursor: crosshair; }
.button {
    background-color: #1ad1ff;
    border: none;
    color: white;
    padding: 8px 16px;
    text-align: center;
    font-size: 16px;
    margin: 4px 2px;
    opacity: 0.6;
    transition: 0.3s;
        border-radius: 15px;


    display: inline-block;
    text-decoration: none;
    cursor: pointer;
```

```
}
.button:hover {opacity: 1}
.cursors {
    display: flex;
    flex-wrap: wrap;
}
</head></style>


<body>
    <h1 style="text-align:center">Draw your letter below to predict:</h1>


        <form action="{{ url_for('predict')}}"method="post">
            <main>
                <div class="left-block"><center>
                    <div class="buttons">
                        <button id="clear"class="button" >Clear</button>
                        <button id="save" class="button" >save</button>


                    </div></center>
                </div>
                <div class="cursors">
                    <div class="crosshair"> <div class="right-block">
```

```
                        <canvas id="paint-canvas" style="background-color:white"
width="100" height="100"></canvas></div>

                    </div>



                </div>

            </main>

        </form>

        <br><br><br><br><br><br><br><br><br><br><br><br><br>

       <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 1440 320">

        <path fill="#0099ff" fill-opacity="1"
d="M0,256L11.4,213.3C22.9,171,46,85,69,90.7C91.4,96,114,192,137,213.3C160
,235,183,181,206,138.7C228.6,96,251,64,274,74.7C297.1,85,320,139,343,149.3
C365.7,160,389,128,411,122.7C434.3,117,457,139,480,133.3C502.9,128,526,96
,549,80C571.4,64,594,64,617,53.3C640,43,663,21,686,32C708.6,43,731,85,754,
117.3C777.1,149,800,171,823,197.3C845.7,224,869,256,891,229.3C914.3,203,9
37,117,960,90.7C982.9,64,1006,96,1029,138.7C1051.4,181,1074,235,1097,240
C1120,245,1143,203,1166,192C1188.6,181,1211,203,1234,192C1257.1,181,128
0,139,1303,122.7C1325.7,107,1349,117,1371,101.3C1394.3,85,1417,43,1429,21
.3L1440,0L1440,320L1428.6,320C1417.1,320,1394,320,1371,320C1348.6,320,13
26,320,1303,320C1280,320,1257,320,1234,320C1211.4,320,1189,320,1166,320
C1142.9,320,1120,320,1097,320C1074.3,320,1051,320,1029,320C1005.7,320,9
83,320,960,320C937.1,320,914,320,891,320C868.6,320,846,320,823,320C800,3
20,777,320,754,320C731.4,320,709,320,686,320C662.9,320,640,320,617,320C5
94.3,320,571,320,549,320C525.7,320,503,320,480,320C457.1,320,434,320,411,
320C388.6,320,366,320,343,320C320,320,297,320,274,320C251.4,320,229,320,
206,320C182.9,320,160,320,137,320C114.3,320,91,320,69,320C45.7,320,23,32
0,11,320L0,320Z"></path>

    </svg>
```

```
<script type="text/javascript">


    window.onload = function () {


   // Definitions
   var canvas = document.getElementById("paint-canvas");
   var context = canvas.getContext("2d");
   var boundings = canvas.getBoundingClientRect();


   // Specifications
   var mouseX = 0;
   var mouseY = 0;
   context.strokeStyle = 'black'; // initial brush color
   context.lineWidth = 5; // initial brush width
   var isDrawing = false;







   // Mouse Down Event
   canvas.addEventListener('mousedown', function(event) {
       setMouseCoordinates(event);
       isDrawing = true;
```

```
    // Start Drawing
    context.beginPath();
    context.moveTo(mouseX, mouseY);
  });
// Mouse Move Event
  canvas.addEventListener('mousemove', function(event) {
    setMouseCoordinates(event);

    if(isDrawing){
      context.lineTo(mouseX, mouseY);
      context.stroke();
    }
  });

  // Mouse Up Event
  canvas.addEventListener('mouseup', function(event) {
    setMouseCoordinates(event);
    isDrawing = false;
  });

  // Handle Mouse Coordinates
  function setMouseCoordinates(event) {
```

```
    mouseX = event.clientX - boundings.left;
    mouseY = event.clientY - boundings.top;
}

// Handle Clear Button
var clearButton = document.getElementById('clear');

clearButton.addEventListener('click', function() {
    context.clearRect(0, 0, 100, 100);
});

// Handle Save Button
var saveButton = document.getElementById('save');

saveButton.addEventListener('click', function(event) {
    var imageName = alert('Is it Ok');
    var canvasDataURL = canvas.toDataURL("image/png");
    var a = document.createElement('a');
    a.href = canvasDataURL;
    a.download = 'iii.png';
    a.click();

});
```

```
};
</script>


</body>
</html>
```

Upload.html:

```
<!DOCTYPE html>
<html>
<head>
<title>Upload</title>
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
</head>
<style>.button {
    background-color: #1ad1ff;
    border: none;
    color: white;
    padding: 8px 16px;
    text-align: center;
    font-size: 16px;
    margin: 4px 2px;
```

```
    opacity: 0.6;

    transition: 0.3s;

        border-radius: 15px;




    display: inline-block;

    text-decoration: none;

    cursor: pointer;

}

.msg{

}

.button:hover {opacity: 1}

</style>

<body>

<div class="wave-container">

<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 1440 320">

    <path fill="#ff5500" fill-opacity="1"
d="M0,224L0,128L51.4,128L51.4,96L102.9,96L102.9,128L154.3,128L154.3,192L
205.7,192L205.7,32L257.1,32L257.1,320L308.6,320L308.6,320L360,320L360,22
4L411.4,224L411.4,64L462.9,64L462.9,224L514.3,224L514.3,32L565.7,32L565.7
,64L617.1,64L617.1,128L668.6,128L668.6,96L720,96L720,192L771.4,192L771.4,
192L822.9,192L822.9,128L874.3,128L874.3,192L925.7,192L925.7,224L977.1,22
4L977.1,96L1028.6,96L1028.6,32L1080,32L1080,320L1131.4,320L1131.4,192L1
182.9,192L1182.9,192L1234.3,192L1234.3,256L1285.7,256L1285.7,96L1337.1,9
6L1337.1,160L1388.6,160L1388.6,192L1440,192L1440,0L1388.6,0L1388.6,0L13
37.1,0L1337.1,0L1285.7,0L1285.7,0L1234.3,0L1234.3,0L1182.9,0L1182.9,0L113
1.4,0L1131.4,0L1080,0L1080,0L1028.6,0L1028.6,0L977.1,0L977.1,0L925.7,0L925
```

```
.7,0L874.3,0L874.3,0L822.9,0L822.9,0L771.4,0L771.4,0L720,0L720,0L668.6,0L668.6,0L617.1,0L617.1,0L565.7,0L565.7,0L514.3,0L514.3,0L462.9,0L462.9,0L411.4,0L411.4,0L360,0L360,0L308.6,0L308.6,0L257.1,0L257.1,0L205.7,0L205.7,0L154.3,0L154.3,0L102.9,0L102.9,0L51.4,0L51.4,0L0,0L0,0Z"></path>

</svg>

<form id="upload-form" action="{{ url_for('upload') }}" method="POST" enctype="multipart/form-data">

<center>

    <strong><h2>Select your drawn File from below:</h2></strong>

    <input id="file-picker" type="file" name="file" accept="image/*" class="button" style="width:250px"multiple>

    <div id="msg" ></div>

        <input type="submit" value="Upload file and Predict English Letter!" id="upload-button" class="button"></center>

</form></div>

</body>

<script>

    $("#file-picker").change(function(){

        var input = document.getElementById('file-picker');

for (var i=0; i<input.files.length; i++)

        {

            var ext= input.files[i].name.substring(input.files[i].name.lastIndexOf('.')+1).toLowerCase()



            if    (ext == 'png')
```

```
                {
                        $("#msg").text("Done..Your File is Supported:)")
                }
                else
                {
                        $("#msg").text("Files are NOT supported")
                        document.getElementById("file-picker").value ="";
                }
            }
        } );
</script>
</html>
```

Prediction.html:

```
<html>
<style>
d {
    font-size:75px;
    position: relative;
    animation: mymove 3s ;
}@keyframes mymove {
    from {left: 0px;}
    to {left: 270px;}
}
```

```
#grad1 {

   height: 100%;

   background-color: red;

   background-image: linear-gradient(to right,    #00ffff, blue, #ffb380);

}

</style>

<body><center>

<div id="grad1"
style="text-align:center;margin:auto;color:#888888;font-size:40px;font-weight:
bold"><br>

  <h1><form >{{prediction_text}}</form></h1>

  <d><span>&#128077;&#127995;</span></d>

</div>

</center>

</body>

</html>
```

CONCLUSION:

Research on automated written language recognition dates back several decades. Today, cleanly machine-printed text documents with simple layouts can be recognized reliably by off-the-shelf OCR software.On the past decade, great efforts are paid to the online hand-drawn content interpretation from handwritten text HWR is a challenging issue because of the large data variability. It is an open research issue to recognize the handwritten words out of scanned documents which is unconstrained.In this document,we have done a review of various HWR models. Depending on different perspectives, models available are reviewed. In future, we try to develop a new HWR model using CRF particularly for Telugu language which will be useful to recognize the Telugu characters from handwriting.the design of humancomputer interfaces based on handwriting is part of a tremendous research effort together with speech recognition, language processing and translation to facilitate communication of people with computer networks. From this perspective, any successes or failure in these fields will have a great impact on the evolution of languages.the design of humancomputer interfaces based on handwriting is part of a tremendous research effort together with speech recognition, language processing and translation to facilitate communication of people with computer networks. From this perspective, any successes or failure in these fields will have a great impact on the evolution of languages.

In this Telugu Hand-Written Letter Recognition and English Letter Prediction project, We just tried for 5 letters.We also did kannada hand-Written letter prediction and its equivalent english letter prediction, here we tried for 10 kannada letters and got accuracy has >97%.We may definitly improve this in future and by next time, we work on recognition of Words.