Title: A Review On Current Handwritten Text Recognition Techniques.

Author : Grace Anulika Eze

1.1 Description of Optical Character Recognition

Handwritten text recognition (HTR) is the ability of systems to convert images of textual data from various input sources both offline (paper based, photos) or online (pen-based computer screen surface). It involves both optical character recognition as well as formatting and segmentation which is a critical stage on handwriting recognition.

However, Optical character recognition (OCR) is a tool used in the identification and extraction of characters such as letters,numbers or symbols from input images by a computer and conversion of this data into a searchable, editable form.

Design of OCR

- Pre-processing
  - Binarization
  - Skew correction
  - Layout analysis
  - Segmentation
  - Script recognition
- Character recognition : two main methods are
  - Matrix matching
  - Feature extraction
- Post-processing

1.2 Libraries/Frameworks

OCR Engines such tesseract, primeOCR, Caere OCR, OCRopus and MathOCR are discussed:

1. Tesseract: It is a popular open source OCR engine which can read a wide variety of image formats. It is free software licensed under Apache 2.0 and can be used to develop programs with libtesseract in C and C++. It can recognize and convert text from over 60+ languages. It accomplishes OCR tasks with a significant success rate but is known to fail if the input image is blur or not pre-processed. The current version (v4) adds a Long term short memory (LSTM) neural network based OCR engine which focuses on line recognition but also still adopts recognition of character patterns as in

previous version. It can be executed from the command line interface or can use a GUI such as OCRfeeder for implementation.

2. OCRopus: It is a free OCR system licensed under Apache v2.0 which was designed for use in large scale conversion of books into digital format, such as Google Books. It also identifies and categorizes regions of interests in input images from documents. It uses statistical language models although recent text recognition are based on recurrent neural networks (LSTM) and do not require a language model so as to train language-independent models which yields good recognition results for several languages at the same time. One future focus is to read documents or scripts which are no longer common (other OCR software don't do this yet).

3. Google Drive OCR : It uses functions of Tesseract or OCRpus and extracts texts from various image source or scanned documents. It can detect and convert over 248 languages with an accuracy of 90% and above. The source code is available on GitHub and is licensed by Apache 2.0. The google OCR can preserve basic character/text formatting such as bold and italic text, font size and type, and line breaks after the OCR in the output text. It is not always successful and other formatting elements like bullets or enumerated lists, tables, text columns, and footnotes or endnotes are usually lost in the process.

4. PrimeOCR : It is a Windows-based OCR engine that reduces OCR error rates by up to 65-80% in comparison to other conventional OCR by implementing "Voting" OCR technology. The Voting in OCR implies that multiple OCR engines have been combined to their solutions. It supports both color and gray-scale images. It can recognize barcodes, recognition of Asian languages characters as well as russian characters and recognition of rotated text. Although it recognizes mostly machine printed texts.

5. Ocrad : This program is based on the feature extraction method and can be used as a stand-alone console application, or as a backend to other programs. It is a free software licensed under GNU General Public License (GPL v2). It reads images of all latin alphabets in pbm (bitmap), pgm (greyscale) or ppm (color) formats and produces text. It also separates blocks of text and follows the layout of the document.

6. Cuneiform : This is a freely distributed open source OCR system which can be used as a library. It performs both single and batch processing of images and documents without altering the layout or font. It is licensed by Freeware/BSD Licenses and can convert over 28 languages. It is written in C and C++. The output formats are in HTML, TeX, txt and so on.

7. MathOCR : This Scientific OCR system is still in its development stages and has several functionalities like image pre-processing, layout analysis, character recognition on various fonts and the added ability to recognize mathematical expressions. It is

written in Java and doesn't depend on external libraries although it can use tesseract or ocrad as backends. It is licensed under the GNU Affero General Public License which is free. It can perform OCR functionalities on several operating systems but not online. It outputs the extracted text in html and latex.

There are also several OCR's which comes in form of a service and considered the current top services based on cost, features, reviews and user experience. The service names are

1. Nuance Omnipage Ultimate : It is one of the top in terms of high speed and accuracy. It can automatically recognize, process and store 120+ languages. It mirrors the layout and design of the scanned document. It is easy to use and has cloud integration. It cost $449.99 and Standard version at $149.99.

2. ABBYY Fine reader : In this software there are many tools like tools to take paper documents from a scanner, make them readable, neatly organized and digitize the documents. It does batch processing and can compare documents, add annotations and comments. It converts over 192 languages into various output formats with an inbuilt spell checker for 14 of them. The cost of the software is $199.99 (Standard version) for a one-time license with OCR conversions and PDF edits. The Corporate version cost $399.99 as it adds the capability for comparing documents and performing automated conversions via a hot folder.

3. ReadIris : It has an easy to use interface according to reviews as well as useful features like fast document processing and filing to several file formats and has an option for text to speech(TTS). It has annotations, comments, password protection for the converted document. It can recognize over 138 languages. It's lowest priced software package is $49.

4. Yunmai OCR SDK : The Sdk are high speed and high accuracy engines which comes also as a service and can be used to recognize documents with 800+ words per page. It recognizes 14+ languages both online and offline and has an advantage in recognizing chinese characters. It is has a proprietary (non free, closed source) license and can be written in languages like Java, C, C++, Objective-C and Pascal. It has several SDKs for various functions like optical character recognition for business cards, Id cards, banking cards(ATM's), passport recognition and so on.

5. TopOCR : It is an interesting service which was designed for automatic document image dewarping with neural warp asides optical character recognition. Neural warp is a new convolutional neural network architecture that can analyse a documents and corrects

page curvature. It claims to do this with a high level of accuracy but asides OCR they don't perform other functions like batch processing etc. It cost about five dollars ($5).

6. Google Drive : It scans any upload for text automatically and the text (or line of text) can be searched either from inside the document or google drive itself. Documents can be scanned directly from the google drive app via the smartphone camera or via a scanner for the computer to begin OCR processing in the drive. It offers several storage capacities for a monthly price.

Also there are several OCR libraries for different programming languages

S/N	Programming language	OCR Libraries	Description
1.	Python	• Pytesseract ○ Licence: GPLv3	It is a python wrapper for Tesseract OCR engine. It recognizes and reads texts from images. It requires support of Python Imaging Library (PIL) and/or support for OpenCV image/NumPy array objects. It can read input images in several formats like jpeg, png, gif, bmp, tiff etc while tesseract-ocr only supports
		• PyOCR ○ Licence: GPLv3+	It is a python wrapper for Tesseract and cuneiform OCR engines with various OCR tools from a Python program. Mainly used on GNU/Linux systems. It supports all the image formats supported by Pillow., Output Format:: text only, bounding boxes

		• Ocrodjvu ◦ License: GPL-2.0-only	It is a library and standalone tool for doing OCR on DjVu documents, wrapping Cuneiform, gocr, ocrad, ocropus and tesseract.
		• Tesserocr ◦ License: MIT	It is a Pillow-friendly, wrapper around the tesseract-ocr API for Optical Character Recognition (OCR) which directly integrates with Tesseract's C++ API using C extensions for python (Cython).
2.	Javascript	• Ocracy ◦ License:	It is a javascript Long Short Term Memory RNN implementation based on OCROPUS.
		• Gocr.js ◦ License: GNU Public License	It an OCR tool that reads images and outputs a text file. Input format : pbm, pgm, png, etc It is also able to recognize and translate barcodes.
		• Tesseract.js ◦ License: Apache License 2.0	Tesseract.js is a javascript library that gets words in almost any language out of images. automatic text orientation and script detection, a simple interface for reading paragraph, word, and character bounding boxes. Tesseract.js can run either in a

			browser and on a server with NodeJS
3.	Golang	● gosseract ○ License: MIT license	It is a Go package for OCR by using Tesseract C++ library.
4.	Ruby	● rtesseract ○ License: MIT license	Ruby library for working with the Tesseract OCR
		● ruby-tesseract ocr ○ License: BSD one clause	A Ruby wrapper library to the tesseract-ocr API. To make this library work you need tesseract-ocr and leptonica libraries and headers and a C++ compiler.
		● ocr_space ○ License: MIT license	It is a free Online OCR for Ruby which converts images to text.

1.3 Algorithms /Model Architecture

There are two major techniques which have been used for the implementation of OCR namely matrix matching and feature extraction.

- ❖ Matrix Matching : This involves comparing an image to a stored character based on pixels. The character from the input image must be isolated from the rest of the image and then compared, for this to work it must also be in a similar font and scale as the stored character. It works well on typewritten texts.

- ❖ Feature extraction: This decomposes characters into "features" like lines, closed loops, line direction, and line intersections. The extraction features reduces the dimensionality of the representation and makes the recognition process computationally efficient. These features are compared with an abstract vector-like representation of a character and then nearest match is chosen.

Several machine learning techniques (such as k-nearest neighbours) and in recent times, neural networks such as Long Term Short Memory (LSTM) RNN,Convolutional Neural Networks (CNN) + LSTM, Gated Recurrent CNN (GRCNN) are being implemented in OCR.

Models

1. Harald Schiedl implemented a neural network approach for handwritten text recognition using Tensorflow. The neural network (NN) was trained on word-images from the IAM dataset and the model consists of 5 convolutional NN (CNN) layers, 2 recurrent NN (RNN) layers and a final Connectionist Temporal Classification (CTC) layer. The model architecture for the HTR system is as follows:

    - The CNN Layers : the input image from the dataset was fed into the CNN layers which then extracts relevant features from the image. There are three operations in every layers, the convolution operation (which applies a filter kernel of size 5×5 in the first two layers and 3×3 in the last three layers to the input), the non-linear RELU function layer and a pooling layer which summarizes image regions and outputs a downsized version of the input. The output feature map had a size of 32x256.
    - RNN : The LSTM implementation of RNN was used because it is able to propagate relevant information through the feature map which contains 256 features per time-step through longer distances and provides more robust training-characteristics. The output feature map for the RNN is mapped to a matrix of size 32×80.

    - CTC : The output of the RNN was then feed to the CTC and the ground truth text and it computes the loss value. While inferring, the CTC is only given the matrix and it decodes it into the final text. Both the ground truth text and the recognized text can be at most 32 characters long.

To improve the recognition accuracy one can either add more CNN layers, replace LSTM by 2D-LSTM, increase dataset-size by applying further (random) transformations to the input images , correct text by replacing recognized word with most similar word if not found in the dictionary, remove formatting from input images.

Link:
https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5
Link to download the IAM handwriting dataset :
http://www.fki.inf.unibe.ch/databases/iam-on-line-handwriting-database

2. Jianfeg wang et al, implemented Gated Recurrent CNN (GRCNN) in combination with a bi-directional LSTM (BLSTM) in recognition of text from natural images and they implemented an unconstrained scene text recognition where each word is recognized without a lexicon unlike previous works on OCR. The architecture for the model consists
    - Feature sequence extraction : They added a gate to a RCNN to control the influence of the extracted pattern in the model to the target pattern hence the Gated Recurrent Convolution Layer (GRCL) is the most important part of their
    - work as it can also weaken or even cut off some irrelevant context information. They stacked the multiple GRCL's with other layers like the Convolution and maxpool layer but no fully connected layer.

    - Sequence modelling : They added a Bi-directional LSTM (BLSTM) to the top of GRCNN which takes in a sequence of inputs from both directions, looks at each element of the sequence and tries to predict the next element of the sequence.

    - Transcription Layer : They used the CTC method (lexicon free recognition)

They tested the accuracy of their model on several datasets like Street View Text (SVT-50) and IIIT5K. They also made comparisons between the accuracy of their model and modern methods like RESNET .

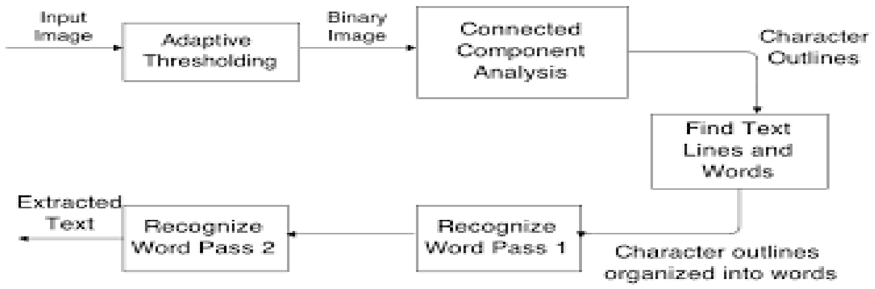
The accuracy was

Models	SVT-50	IIIT5K-50
GRCNN-BLSTM	96.3%	98.0%
RESNET-BLSTM	96.0%	97.5%
CRNN (By B. Shi et al)	96.4%	97.6%

Their model seemed to outperform many others state of the art models created by others when compared with & datasets.

https://papers.nips.cc/paper/6637-gated-recurrent-convolution-neural-network-for-ocr.pdf

3. Tesseract-OCR architecture



The architecture of the popular tesseract-OCR as shown above consist of

- Adaptive thresholding : Here input images of grayscale or color are converted binary images by selecting individual thresholds for each pixel based on the range of intensity values in its local neighborhood.
- Connected component analysis: This is used to extract character outlines which are in in connected regions in the binary image
- Line finding algorithm : This finds text lines and words so that a skewed page can be recognized without having to de-skew, thus saving loss of image quality.

- Two pass Recognition method : For pass 1, it tries to recognize each word and each satisfactory word is then passed to an adaptive classifier as training data. The classifier then attempts to accurately recognize the remaining text so that were not recognized well enough are recognized again.

4. Zhanzhan Cheng et al implemented a Focusing Attention Network (FAN) in scene text recognition. They tried to propose a the model as a way to battle the “attention drift problem which is when attention models can’t get accurate alignment between feature areas and targets for such images in low quality images. Their method used a focusing attention mechanism to automatically draw back the drifted attention to the text. FAN consists of two major components:
    - Attention network (AN): It recognizes character targets as in the existing methods
    - Focusing network (FN): It adjusts attention by evaluating whether AN pays attention properly on the target areas in the images.

First they used a CNN-BLSTM (ResNet-based network) to transform the image into a sequence of feature vectors which corresponds to a region in the image. The AN is then used to extract features that generate the alignment factors and glimpse vectors, with which FN focuses the attention of AN on the proper target character regions in the images. Then, the LSTM is used to generate the target characters based on the glimpse vectors and the history of target characters. They tested on several dataset like IIIT5k and SVT.

Models	SVT-50	IIIT5K-50

FAN	97.1%	99.3%
AN (Z.cheng)	96.0%	97.5%
CRNN (By B. Shi et al)	96.4%	97.6%

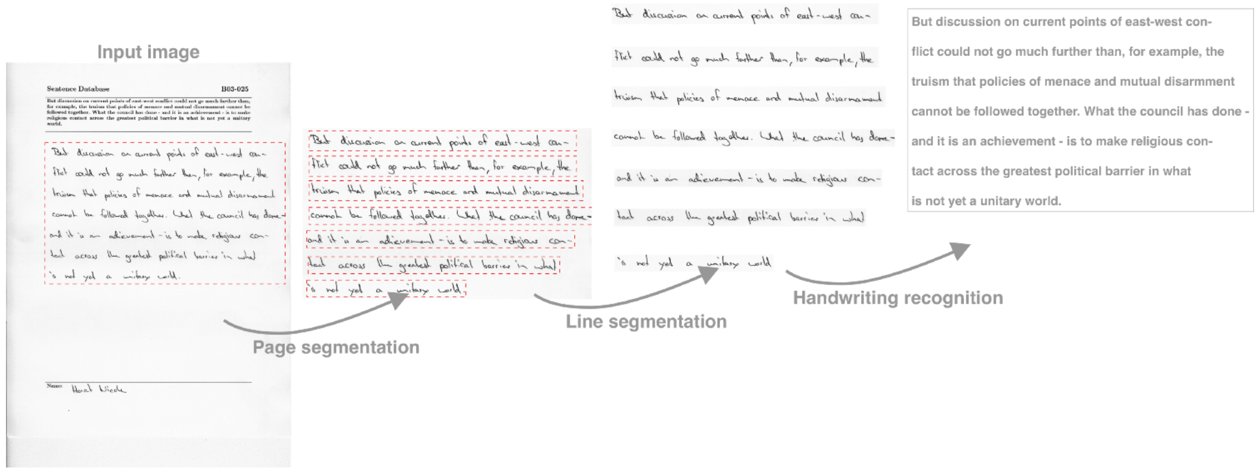
https://arxiv.org/abs/1709.02054

5. Dropbox OCR (Word Deep Net) : The model architecture is
   - Stack of convolutional layers
   - Stack of bidirectional LSTM layers
   - Connectionist Temporal Classification(CTC) output layer

They used Batch Normalization where they deemed appropriate and also are investigating the use of Generative Adversarial Networks (GANs) to generate realistic data for training.

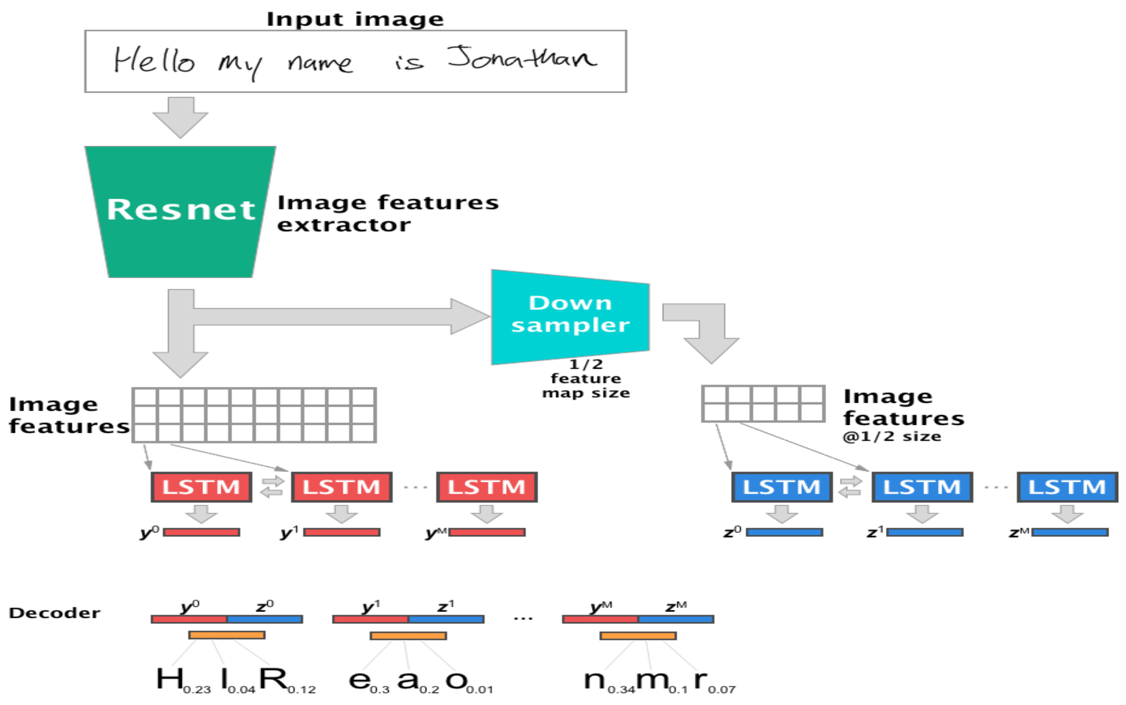
https://blogs.dropbox.com/tech/2017/04/creating-a-modern-ocr-pipeline-using-computer-vision-and-deep-learning/

6. Jonathan Chung and Thomas D. also implemented a neural network approach for handwritten text recognition using MXNET Gluon. His model which consisted of CNN layers , BiLSTM as well as a CTC layer was trained on word-images from the IAM dataset.



But discussion on current points of east-west con-
flict could not go much further than, for example, the
truism that policies of menace and mutual disarmment
cannot be followed together. What the council has done -
and it is an achievement - is to make religious con-
tact across the greatest political barrier in what
is not yet a unitary world.

He used a a pre-trained res-net as a image feature extractor and he extended his CNN-BiLSTM by adding a multiple downsamples of the image features as a means to assist in the recognition of images of handwritten text that vary in size .



Then from the output of the model, the probability distribution over the characters for each vertical slice of the image is predicted fed using a decoder. Next, they experimented with several language model (e.g greedy search, lexicon searching, and beam searching + lexicon searching + language model) to extract the most probably sentence from the matrix of probabilities.

https://github.com/ThomasDelteil/HandwrittenTextRecognition_MXNet

https://medium.com/apache-mxnet/handwriting-ocr-handwriting-recognition-and-language-modeling-with-mxnet-gluon-4c7165788c67

1.4 Input Data Limitations

Although OCR is viewed as a good data extraction method , there are still some challenges in the data extraction process

1. Training OCR systems to reduce strike out errors (i.e when writer strikes-out a wrong/inadequate word and writes the proper word next to it). http://cdn.iiit.ac.in/cdn/cvit.iiit.ac.in/SSDA/slides/Jaipur%20Strike_out_BBChaudhuri.pdf

2. The OCR may have problems detecting handwritten text ,textual data in special font type or document formatting (like bold,underline or italics) is lost during processing.
3. Several characters or words may be unavailable since OCR may not convert characters with very large or very small font sizes.
4. Incorrect recognition or loss of special characters in OCR for various languages except the right OCR language pack is used manually.
5. Trouble recognizing Mathematical formulas.
6. OCR has trouble differentiating case sensitive letters/words such as p and P or cat and CAT.
7. The OCR may not be able to convert non-textual glyphs like logos, or map symbols thus leaving them invisible for text analytics or text-based retrieval.
8. The OCR systems may detect some text and assume its not needed or ignore text in images or from headers or footers thus leaving incomplete text.
9. OCR is unable to work in ancient manuscripts and writing style and in those documents where the ink is faded in some part of the texts.
10. Collecting enough data to train current deep learning models for OCR in order to achieve great accuracy in recognition is tedious and time consuming because the train data has to be collected and labeled manually (thus the current trend of using GAN's can help in generating realistic train data) .

Accuracy

1. Low accuracy on blur, skewed or color images
2. OCR gives better accuracy when working with images of simple text images with uniform layouts or short strings of text.
3. Special fonts or specific characters influence the accuracy of the OCR.

Other Research methods On Handwritten Datasets				
Author	Dataset	Model	Accuracy	Links
Theodore Bluche et al (2016)	IAM Database RIMEs database (french)	● GCRNN + MDLSTM +CTC ● Attention based model	-	http://www.tbluche.com/files/icdar17_gnn.pdf
Adeline Granet (2018)	Italian Comedy (CI) dataset	FCN-BLSTM-CTC	84.2% CRR for RM	http://www.scitepress.org/Pa

	RIMES (RM) dataset Los Esposalles (ESP) dataset			pers/2018/65988/65988.pdf
Curtis Wigington et al (2018)	ICDAR2017 HWR competition dataset Dataset link: https://zenodo.org/record/835489#.XH0A3IMzYWI	SFR Model ● the Start of Line (SOL) network, ● the Line Follower (LF) network, ● and the Handwriting Recognition (HWR) network. Region Proposal Network (RPN) which detects the start of the text line. CNN-LSTM for the HWR aspect.	-	http://openaccess.thecvf.com/content_ECCV_2018/papers/Curtis_Wigington_Start_Follow_Read_ECCV_2018_paper.pdf
Rushikesh Laxmikant Kulkarni (2017)	HASYv2 dataset	SVM classifier to train the dataset ● HOG for text detction	96.56%	http://www.ijarcsms.com/docs/paper/volume5/issue4/V5I4-0018.pdf

1.5 Text Detection

Some of the current state-of-the-art techniques being used in text detection are

- Maximally Stable Extremal Regions (MSER) : It was used to handcraft features and used to detect connected areas on with text on the images.
  - For each threshold, compute the connected binary regions.
  - Compute a function, area A(i), at each threshold value i.
  - Analyze this function for each potential region to determine those that persist with similar function value over multiple thresholds.

- ➢ Histogram of Oriented Gradients (HOG) : It is also used in describing/extracting the features in all positions of an image (or region of interest in the image) and commonly used for object detection.
    - It calculates the directional change(gradient values) in the intensity of the image in order to get the shape of structures.
    - It then puts this in a bin
    - and once all locations in the image has been and binned it then tries to determine where the edges are.

- ➢ Canny Edge Detection Algorithm : It is used in extracting important structural information from images and reduces the amount of data to be processed.

    - Apply Gaussian filter to smooth the image in order to remove the noise
    - Find the intensity gradients of the image
    - Apply non-maximum suppression to get rid of spurious response to edge detection
    - Apply double threshold to determine potential edges
    - Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

- ➢ Scale Invariant Feature Transform (SIFT) : It is a hand engineered and doesn't learn the features by itself . It is based on first order gradients and is evaluated around scale invariant feature points which are obtained using the difference of gaussian (DoG) key point detector. It doesn't require much processing power and memory to process features of images and very relevant for identification tasks.

- ➢ Convolutional Neural Networks (CNN) : This involves several convolutional operations which involves feature detectors which can be trained and can learn to adapt to the task at hand. They can detect texts with high accuracy as they can learn low-level features from just training. The networks make uses of hierarchical layer-wise representation learning Thus one can optimizes feature extraction unlike other methods.

<u>Research papers on text detection for HWR</u>

Author	Dataset	Model	Links
Lorenzo Quir´os (2018)	-Oficio de Hipotecas de Girona (OHG) dataset - Competition on Baseline Detection in	Document Layout Analysis ● Pixel level classification, for zones and baselines (via ANN) ● Zone segmentation and baseline	https://arxiv.org/pdf/1806.08852.pdf

	Archival Documents (cBAD) dataset - Bozen dataset	detection	
Sourav Ghosh et al (2017/2018)	Bangla handwritten documents	Local Binary Pattern (LBP) and its variants.	https://www.mdpi.com/2313-433X/4/4/57
Md. Rabiul Islam et al (2016)	ICDAR 2011 dataset	Enhanced MSERs	https://ieeexplore.ieee.org/document/7760054

1. Chen et al. used Convolutional Autoencoders for page segmentation of handwritten documents. Then the feature representations are fed into an SVM classifier to learn the segmentations of the document.

   http://cyber.sci-hub.tw/MTAuMTEwOS9pY2Rhci4yMDE1LjczMzM5MTQ=/10.1109%40ICDAR.2015.7333914.pdf

2. Jonathan chung worked on Page Segmentation and tried to fit a bounding box around the handwritten portion of the document from the IAM dataset. There are two methods to obtain the bounding box
   - MSERs algorithm approach which they tried to identify continuous regions of text. For each iteration, the bounding boxes were expanded by a fraction in all directions. Then the boxes which overlap a certain percentage are merged.
   - Deep CNN approach

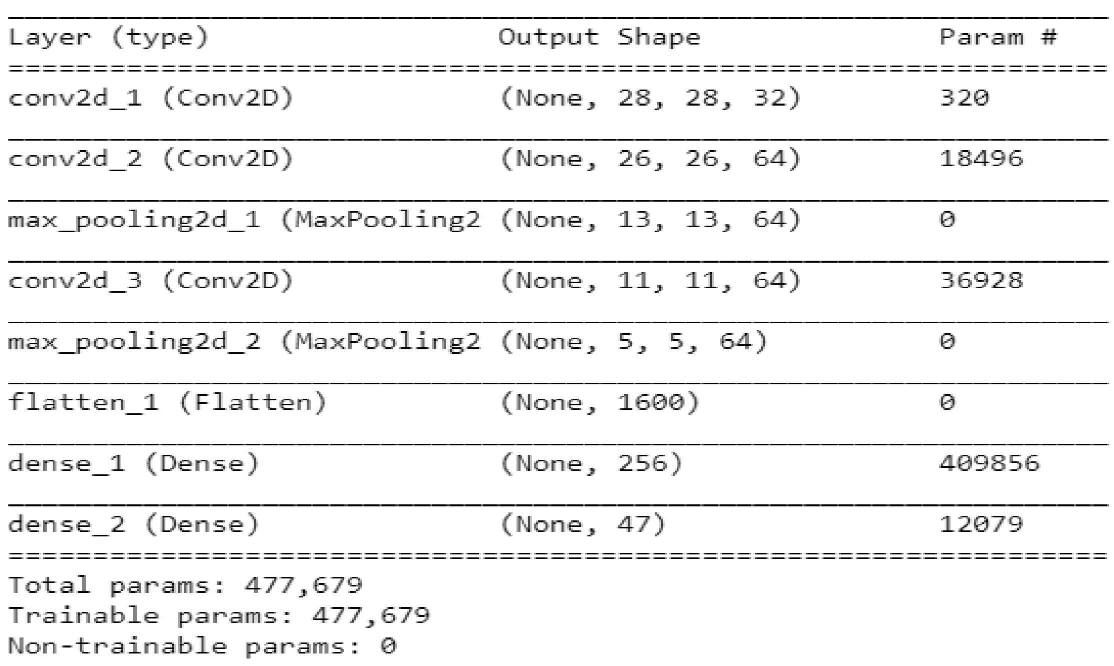
https://medium.com/apache-mxnet/page-segmentation-with-gluon-dcb4e5955e2

2. Implementing simple models as a Practical Task

We were asked to find any handwriting dataset and choose modern architecture to implement a simple handwritten OCR model. For this task I chose :

1. The EMNIST character dataset and
2. The IAM Handwriting Dataset

2.1 The EMNIST dataset is an extended set of the MNIST handwritten character digits which contains both digits and alphabets. It contains 62 classes with 0-9 digits and A-Z characters in both uppercase and lowercase. For this dataset, I read in the data, split the dataset and reshaped it. I decided to implement a simple convolutional neural network using the Keras library with the sequential model. The model consisted of three (3) convolutional layers with ReLU activations which extracts the features of the image, two(2) max pooling layers which reduces the spatial size of the input coming from the convolution layer and two(2) dense layers.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 26, 26, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 11, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               409856
_________________________________________________________________
dense_2 (Dense)              (None, 47)                12079
=================================================================
Total params: 477,679
Trainable params: 477,679
Non-trainable params: 0
```

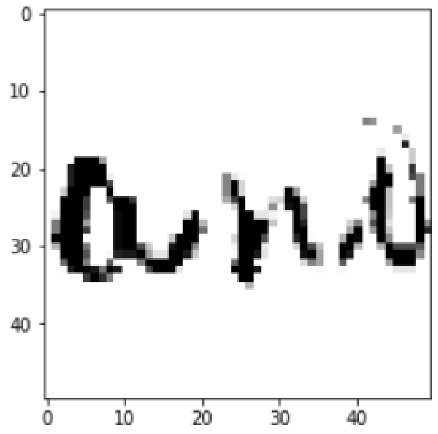
After compiling the model with a categorical cross-entropy loss and an adam optimizer, the model was fitted and after some number of iterations the loss was 0.2918 while the accuracy of the model was 89.09% on the train data.

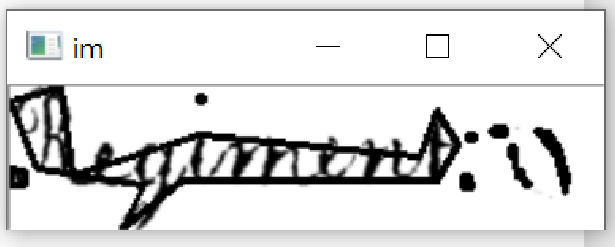
```
Epoch 1/3
101519/101519 [==============================] - 270s 3ms/step - loss: 0.5780
- acc: 0.8123
Epoch 2/3
101519/101519 [==============================] - 272s 3ms/step - loss: 0.3460
- acc: 0.8758
Epoch 3/3
101519/101519 [==============================] - 270s 3ms/step - loss: 0.2918
- acc: 0.8909
```

The model was then evaluated on the test data, the loss was 0.3682 and the accuracy was 87.81% .

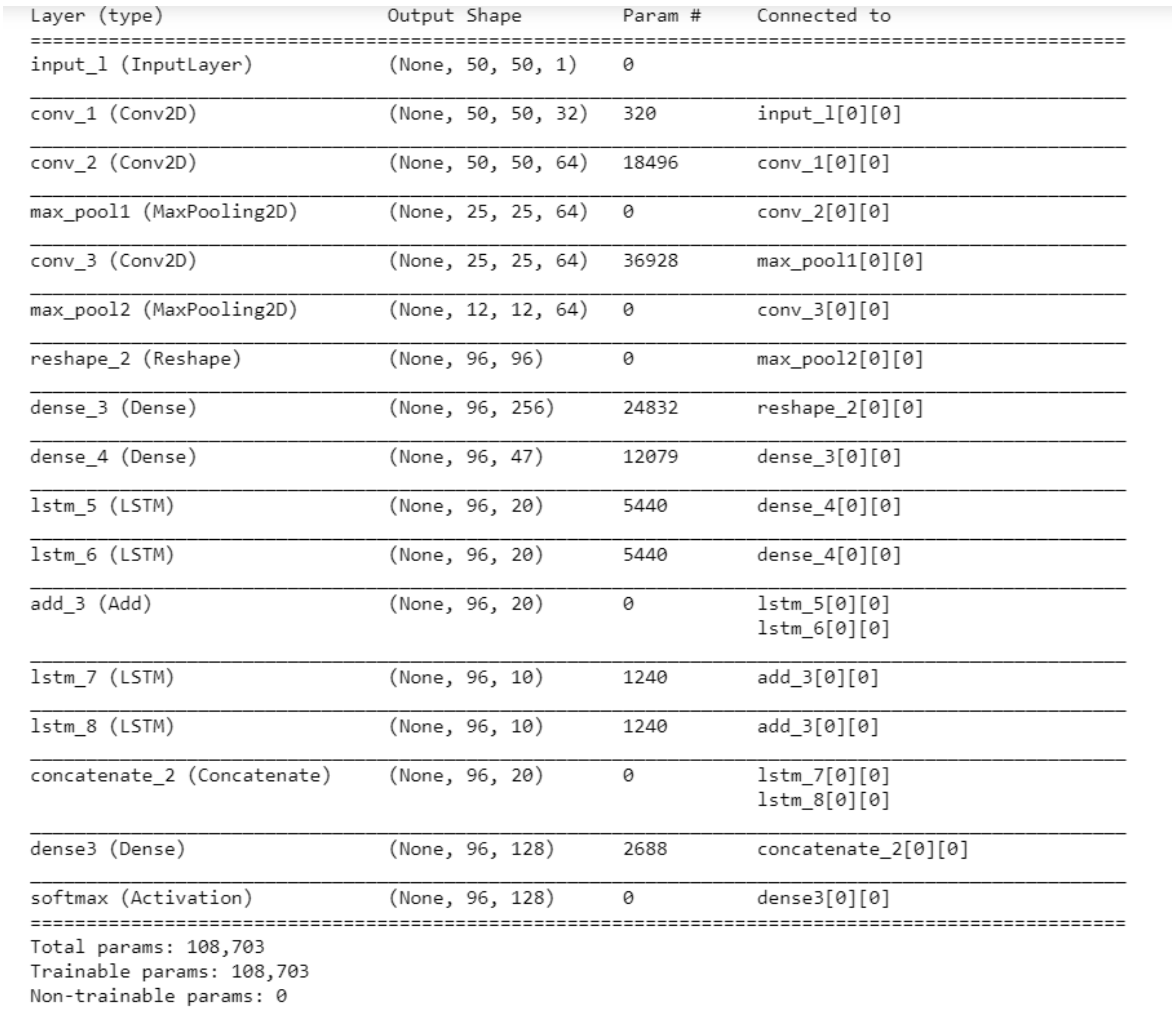
2.2 The IAM handwriting dataset contains forms of handwritten english text such as 0-9 and A-Z in both and has images of both sentence lines and also images of just single word. Firstly I read in the ground truth text, then read the words into list and converted each character in the word to its ASCII equivalent to make it easier for the model to recognize the sequence of words. I also read in the images, did some preprocessing like converting to grayscale and resizing the image.

I also tried to visualize the image data and also detect the words in the images





I decided to implement a simple model using the Keras library with the Functional API model. The model consisted of three (3) convolutional layers with ReLU activations which extracts the features of the image, two(2) max pooling layers which reduces the spatial size of the input coming from the convolution layer and two(2) dense layers. After extracting the features of the



```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 50, 50, 1)    0
__________________________________________________________________________________________________
conv_1 (Conv2D)                 (None, 50, 50, 32)   320         input_1[0][0]
__________________________________________________________________________________________________
conv_2 (Conv2D)                 (None, 50, 50, 64)   18496       conv_1[0][0]
__________________________________________________________________________________________________
max_pool1 (MaxPooling2D)        (None, 25, 25, 64)   0           conv_2[0][0]
__________________________________________________________________________________________________
conv_3 (Conv2D)                 (None, 25, 25, 64)   36928       max_pool1[0][0]
__________________________________________________________________________________________________
max_pool2 (MaxPooling2D)        (None, 12, 12, 64)   0           conv_3[0][0]
__________________________________________________________________________________________________
reshape_2 (Reshape)             (None, 96, 96)       0           max_pool2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 96, 256)      24832       reshape_2[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 96, 47)       12079       dense_3[0][0]
__________________________________________________________________________________________________
lstm_5 (LSTM)                   (None, 96, 20)       5440        dense_4[0][0]
__________________________________________________________________________________________________
lstm_6 (LSTM)                   (None, 96, 20)       5440        dense_4[0][0]
__________________________________________________________________________________________________
add_3 (Add)                     (None, 96, 20)       0           lstm_5[0][0]
                                                                 lstm_6[0][0]
__________________________________________________________________________________________________
lstm_7 (LSTM)                   (None, 96, 10)       1240        add_3[0][0]
__________________________________________________________________________________________________
lstm_8 (LSTM)                   (None, 96, 10)       1240        add_3[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 96, 20)       0           lstm_7[0][0]
                                                                 lstm_8[0][0]
__________________________________________________________________________________________________
dense3 (Dense)                  (None, 96, 128)      2688        concatenate_2[0][0]
__________________________________________________________________________________________________
softmax (Activation)            (None, 96, 128)      0           dense3[0][0]
==================================================================================================
Total params: 108,703
Trainable params: 108,703
Non-trainable params: 0
```

images, I added two (2) Bidirectional Long Term Short Memory(LSTM’s) then sent it to a final dense layer.

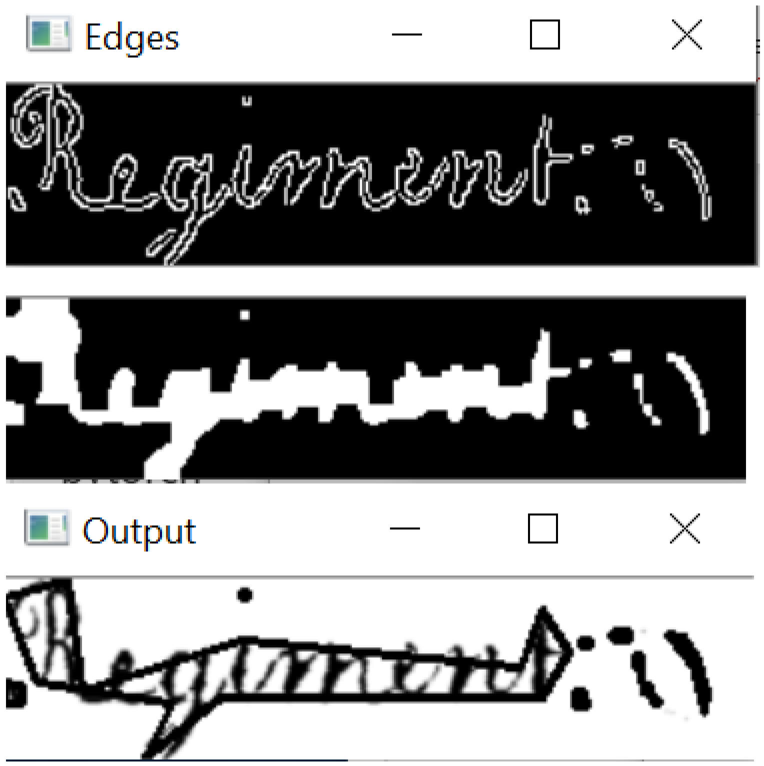
The convolutional layers (CNN) extract a sequence of features and recurrent layers (bidirectional LSTM) to propagate information through this sequence in a forward and backward manner. It outputs character-scores for each sequence-element, which simply is represented by a matrix.
The loss value is calculated to train the NN and decode the matrix to get the text contained in the input image. The CTC loss function is feed the output matrix of the NN and the corresponding ground-truth (GT) text. It tries all possible alignments of the GT text in the image and takes the sum of all scores. This way, the score of a GT text is high if the sum over the alignment-scores has a high value. The NN outputs a matrix containing a score for each character at each time-step. The loss is calculated by summing up all scores of all possible alignments of the GT text, this way it does not matter where the text appears in the image.

For the detection this was the code:

```
import cv2
#reading the image  #270-01
image                                                                                    =
cv2.pyrDown(cv2.imread('C:/Users/ghrac/Documents/Internship/washingtondb-v1.0/washington
db-v1.0/data/line_images_normalized/272-05.png', cv2.IMREAD_UNCHANGED))

edged = cv2.Canny(image, 10, 250)
cv2.imshow("Edges", edged)
cv2.waitKey(0)

#applying closing function
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (7, 7))
closed = cv2.morphologyEx(edged, cv2.MORPH_CLOSE, kernel)
cv2.imshow("Closed", closed)
cv2.waitKey(0)

#finding_contours
(_,cnts,        _)        =        cv2.findContours(closed.copy(),        cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

for c in cnts:
	peri = cv2.arcLength(c, True)
	approx = cv2.approxPolyDP(c, 0.02 * peri, True)
	cv2.drawContours(image, [approx], -1, (0, 255, 0), 2)
cv2.imshow("Output", image)
cv2.waitKey(0)


(_,cnts1,        _)        =        cv2.findContours(edged.copy(),        cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
idx = 0
for c1 in cnts1:
	x,y,w,h = cv2.boundingRect(c1)
	if w>50 and h>50:
		idx+=1
		new_img=image[y:y+h,x:x+w]
		cv2.imwrite(str(idx) + '.png', new_img)
cv2.imshow("im",image)
cv2.waitKey(0)

print("Found %d objects." % len(cnts))
```

```
for (i, c) in enumerate(cnts):
    print("\tSize of contour %d: %d" % (i, len(c)))
```

The outputs are:



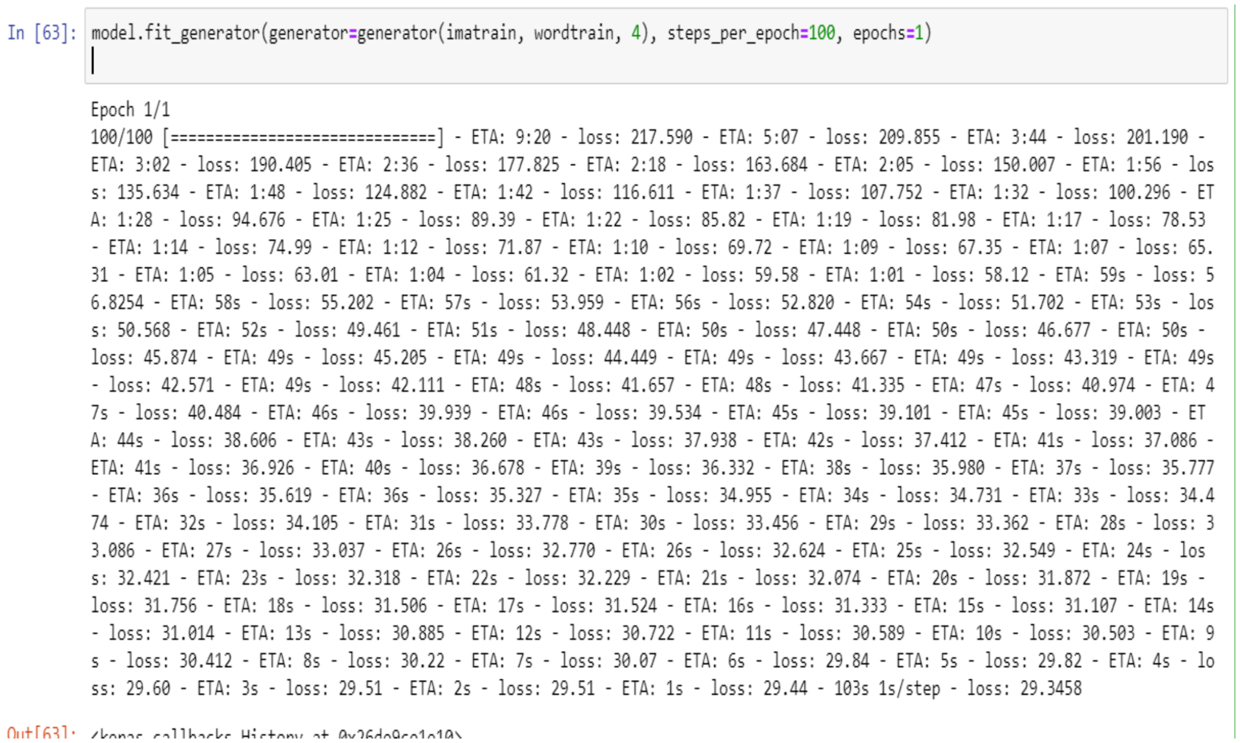
<u>For the CNN-Bi_LSTM-CTC model :</u>

```
#second approach

input_l = Input(name='input_l', shape=(50,50,1), dtype='float32')
net = Conv2D(32, kernel_size=(3,3), name='conv_1', padding='same', activation='relu')(input_l)
net = Conv2D(64, kernel_size=(3,3), name='conv_2', padding='same', activation='relu')(net)
net = MaxPooling2D((2, 2), name='max_pool1')(net)
net = Conv2D(64, kernel_size=(3,3), name='conv_3', padding='same', activation='relu')(net)
net = MaxPooling2D((2, 2), name='max_pool2')(net)
net = Reshape(target_shape=(96,96))(net)
net = Dense(256,activation='relu')(net)
net = Dense(47,activation='relu')(net)
```

```
lstm_f1 = LSTM(20, return_sequences=True, activation='tanh', inner_activation='sigmoid',
input_shape=(47, ))(net)
lstm_b1 = LSTM(20, return_sequences=True, activation='tanh', inner_activation='sigmoid',
input_shape=(47, ), go_backwards=True)(net)
merged_lstm1 = add([lstm_f1, lstm_b1])

lstm_f2 = LSTM(10, return_sequences=True, activation='tanh', inner_activation='sigmoid'
)(merged_lstm1)
lstm_b2 = LSTM(10, return_sequences=True, activation='tanh', inner_activation='sigmoid',
go_backwards=True)(merged_lstm1)
merged_lstm2 = add([lstm_f2, lstm_b2])

inner = Dense(128, kernel_initializer='he_normal', name='dense3')(concatenate([lstm_f2,
lstm_b2]))  ## transforms LSTM output to character activations
y_pred = Activation('softmax', name='softmax')(inner)
Model(inputs=input_l, outputs=y_pred).summary()
```

```
def generator(imatrain,wordtrain,batch_size):
    max_len=100
    j=0
    #print (imatrain.shape, wordtrain.shape)
    while True:
        for cbatch in range(0, imatrain.shape[0], batch_size):
            j=j+1
            x_batch=imatrain[cbatch:(cbatch + batch_size),:,:]
            y_batch=wordtrain[cbatch:(cbatch + batch_size)]
            size=x_batch.shape[0]
            labels = np.ones([size, max_len])
            input_length = np.zeros([size, 1])
            label_length = np.zeros([size, 1])
            source_str = []
           # print (cbatch)
            for i in range (x_batch.shape[0]):
                labels[i, :len(y_batch[i])] = y_batch[i]
                input_length[i] = x_batch.shape[1]
                label_length[i] =len(y_batch[i])
                source_str.append('')
            inputs_again = {'input_l': x_batch,
                   'the_labels': labels,
                   'input_length': input_length,
                   'label_length': label_length,
            'source_str': source_str  # used for visualization only
            }
        outputs = {'ctc': np.zeros([size])}
        yield(inputs_again,outputs)



def ctc_lambda_func(args):
    y_pred, labels, input_length, label_length = args
    # the 2 is critical here since the first couple outputs of the RNN
    # tend to be garbage:
    y_pred = y_pred[:, 2:, :]
    return K.ctc_batch_cost(labels, y_pred, input_length, label_length)



labels = Input(name='the_labels', shape=[100], dtype='float32')


input_length = Input(name='input_length', shape=[1], dtype='int64')
label_length = Input(name='label_length', shape=[1], dtype='int64')
#  CTC loss is implemented in a lambda layer
loss_out = Lambda(ctc_lambda_func, output_shape=(1,), name='ctc')([y_pred, labels,
input_length, label_length])

# clipnorm seems to speeds up convergence
sgd = SGD(lr=0.02, decay=1e-6, momentum=0.9, nesterov=True, clipnorm=5)

#model = Model(inputs=[input_l, labels, input_length, label_length], outputs=loss_out)
model = Model(inputs=[input_l, labels, input_length, label_length], outputs=loss_out)



# the loss calc occurs elsewhere, so use a dummy lambda func for the loss
model.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=sgd)
```

The output after fitting the model is :



```
In [63]: model.fit_generator(generator=generator(imatrain, wordtrain, 4), steps_per_epoch=100, epochs=1)

         Epoch 1/1
         100/100 [==============================] - ETA: 9:20 - loss: 217.590 - ETA: 5:07 - loss: 209.855 - ETA: 3:44 - loss: 201.190 -
         ETA: 3:02 - loss: 190.405 - ETA: 2:36 - loss: 177.825 - ETA: 2:18 - loss: 163.684 - ETA: 2:05 - loss: 150.007 - ETA: 1:56 - los
         s: 135.634 - ETA: 1:48 - loss: 124.882 - ETA: 1:42 - loss: 116.611 - ETA: 1:37 - loss: 107.752 - ETA: 1:32 - loss: 100.296 - ET
         A: 1:28 - loss: 94.676 - ETA: 1:25 - loss: 89.39 - ETA: 1:22 - loss: 85.82 - ETA: 1:19 - loss: 81.98 - ETA: 1:17 - loss: 78.53
         - ETA: 1:14 - loss: 74.99 - ETA: 1:12 - loss: 71.87 - ETA: 1:10 - loss: 69.72 - ETA: 1:09 - loss: 67.35 - ETA: 1:07 - loss: 65.
         31 - ETA: 1:05 - loss: 63.01 - ETA: 1:04 - loss: 61.32 - ETA: 1:02 - loss: 59.58 - ETA: 1:01 - loss: 58.12 - ETA: 59s - loss: 5
         6.8254 - ETA: 58s - loss: 55.202 - ETA: 57s - loss: 53.959 - ETA: 56s - loss: 52.820 - ETA: 54s - loss: 51.702 - ETA: 53s - los
         s: 50.568 - ETA: 52s - loss: 49.461 - ETA: 51s - loss: 48.448 - ETA: 50s - loss: 47.448 - ETA: 50s - loss: 46.677 - ETA: 50s -
         loss: 45.874 - ETA: 49s - loss: 45.205 - ETA: 49s - loss: 44.449 - ETA: 49s - loss: 43.667 - ETA: 49s - loss: 43.319 - ETA: 49s
         - loss: 42.571 - ETA: 49s - loss: 42.111 - ETA: 48s - loss: 41.657 - ETA: 48s - loss: 41.335 - ETA: 47s - loss: 40.974 - ETA: 4
         7s - loss: 40.484 - ETA: 46s - loss: 39.939 - ETA: 46s - loss: 39.534 - ETA: 45s - loss: 39.101 - ETA: 45s - loss: 39.003 - ET
         A: 44s - loss: 38.606 - ETA: 43s - loss: 38.260 - ETA: 43s - loss: 37.938 - ETA: 42s - loss: 37.412 - ETA: 41s - loss: 37.086 -
         ETA: 41s - loss: 36.926 - ETA: 40s - loss: 36.678 - ETA: 39s - loss: 36.332 - ETA: 38s - loss: 35.980 - ETA: 37s - loss: 35.777
         - ETA: 36s - loss: 35.619 - ETA: 36s - loss: 35.327 - ETA: 35s - loss: 34.955 - ETA: 34s - loss: 34.731 - ETA: 33s - loss: 34.4
         74 - ETA: 32s - loss: 34.105 - ETA: 31s - loss: 33.778 - ETA: 30s - loss: 33.456 - ETA: 29s - loss: 33.362 - ETA: 28s - loss: 3
         3.086 - ETA: 27s - loss: 33.037 - ETA: 26s - loss: 32.770 - ETA: 26s - loss: 32.624 - ETA: 25s - loss: 32.549 - ETA: 24s - los
         s: 32.421 - ETA: 23s - loss: 32.318 - ETA: 22s - loss: 32.229 - ETA: 21s - loss: 32.074 - ETA: 20s - loss: 31.872 - ETA: 19s -
         loss: 31.756 - ETA: 18s - loss: 31.506 - ETA: 17s - loss: 31.524 - ETA: 16s - loss: 31.333 - ETA: 15s - loss: 31.107 - ETA: 14s
         - loss: 31.014 - ETA: 13s - loss: 30.885 - ETA: 12s - loss: 30.722 - ETA: 11s - loss: 30.589 - ETA: 10s - loss: 30.503 - ETA: 9
         s - loss: 30.412 - ETA: 8s - loss: 30.22 - ETA: 7s - loss: 30.07 - ETA: 6s - loss: 29.84 - ETA: 5s - loss: 29.82 - ETA: 4s - lo
         ss: 29.60 - ETA: 3s - loss: 29.51 - ETA: 2s - loss: 29.51 - ETA: 1s - loss: 29.44 - 103s 1s/step - loss: 29.3458

Out[63]: <keras.callbacks.History at 0x26de9ce1e10>
```

<u>THANK YOU FOR THE OPPORTUNITY.</u>