MANIPAL
ACADEMY of HIGHER EDUCATION
(Deemed to be University under Section 3 of the UGC Act, 1956)

MANIPAL SCHOOL OF INFORMATION SCIENCES
(A Constituent unit of MAHE, Manipal)

Image to text to speech conversion with intonation using machine learning for regional language

Reg. Number	Name	Branch
201046012	Pooja C A	Big Data & Data Analytics
201046029	Karthik Ballullaya MK	Big Data & Data Analytics
201046042	K. Viswateja	Big Data & Data Analytics

Under the guidance of

Prof. Dr. Madhushankara. M

Manipal School of Information Sciences,
MAHE, MANIPAL

10/12/2020

MANIPAL SCHOOL OF INFORMATION SCIENCES
MANIPAL
(A constituent unit of MAHE, Manipal)

DECLARATION

We declare that this mini project, submitted for the evaluation of course work of Mini Project to Manipal School of Information Sciences, is implementing Image to text to speech conversion with intonation using machine learning for regional language conducted under the supervision conducted under the supervision of my guide Dr. Madhushankara. M and panel members Mr. Shridhar Nayak and Dr. Madhushankara M. References, help and material obtained from other sources have been duly acknowledged.

ABSTRACT

The objective of this project is to convert images to text and text to speech with intonation using machine learning for a regional language by classifying characters written in Kannada, a south Indian language. The characters are extracted from written documents, segmented, and processed using various image processing techniques such as contrast normalization, denoising, thinning etc. using numpy and OpenCV. The dataset used is a combination of the Chars74k dataset and a custom-made dataset. Classifiers have been implemented based on K Nearest Neighbors, Support Vector Machines, Inception V3 and Convolutional Neural Networks using OpenCV and Keras.

Keywords

Image processing, OCR, Data Agumentation, k-NN, SVM, Inception V3 and C-NN, Transfer Learning, Tensorflow, OpenCV and Keras.

Contents

LIST OF FIGURES

ABBREVIATIONS

CNN	Convolution Neural Network
SVM	Support Vector Machine
k-NN	k-Nearest Neighbors

1. Introduction

The objective of this project is to convert images to text and text to speech with intonation using machine learning for a regional language by classifying characters written in Kannada, a south Indian language. Kannada is a Dravidian language spoken predominantly in the Indian state of Karnataka. Kannada is written in a non-Latin script and has forty-nine phonemic letters, divided into three groups: 13 vowels, 34 consonants and 2 letters that are neither consonants nor vowels. Compared to Latin character recognition, Kannada character detection is a much more complicated problem because of the larger character set and similarity between characters. Most research uses only very well written isolated characters or handwritten documents in ideal conditions. Recognition of characters in a complete handwritten document poses a much harder problem due to the varying width of characters and the presence of *kagunita* and *ottakshara* which lead to uneven spacing between characters and sentences.

Optical character recognition (OCR) is the process of converting handwritten or printed text into a digital form. OCR can be either online or offline. Offline OCR works on already printed or written text while online OCR works on characters being written using a special optical pen and an electronic pad. OCR in Indian languages has been an interesting point of research due to the vast character set and its complexity. The accuracy of the OCR depends largely on the steps prior to the recognition i.e. segmentation, preprocessing. Methods have been proposed to segment a document into sentences, words and then finally characters for isolated character recognition. Fig 1 shows the various steps used as a flowchart. Further methods have been used to preprocess the image in such a way that reduces the memory needed by the models to process the characters.

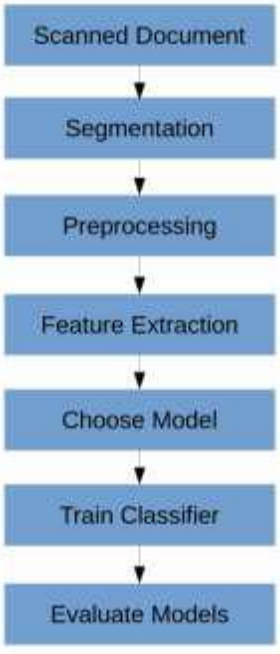

Fig.1. Various stages in OCR

2. Material and methods

DATASET

One of the datasets used is the Chars74K dataset [1]. It consists of a collection of images that belong to 657+ classes. Each image consists of a single character or a combination of characters. Each class consists of 25 handwritten characters. Since a Deep Learning approach has been used, the dataset needs to be expanded. This is done using handwritten text provided by 30 native volunteers. To further expand the dataset, augmentation techniques is used. This custom dataset is referred to as PSCube.

SEGMENTATION

A. Line segmentation

The entire document is segmented into individual lines. Contour detection and bounding boxes have been used to implement this. The image is first dilated to compensate for the pixels lost while scanning the image or in case the pen stroke is not regular. Next, the external contours are detected across the entire image. The horizontal, vertical, and diagonal segments of the detected contours are then compressed to leave only their endpoints which eventually represents a rectangle around each line. The contours are then sorted and for each contour, a bounding box is drawn around each line. The area covered by the bounded box is then extracted and stored as a separate file thus leading to a series of segmented lines.

B. Word segmentation

After line segmentation, each word in the line is extracted in a serial manner. The two approaches considered are Blob detection and Dilation. Though blob detection method is more accurate, it is a very slow approach. Hence, dilation method is chosen as it is simpler and comparatively faster.

C. Character segmentation

There are several approaches that have been tested and experimented in the past for the same. The methods considered are Vertical Projection Profile and Bounding Box Analysis. Bounding box analysis method is chosen for character segmentation as it provides results with considerably less computation cost.

PREPROCESSING

Preprocessing is the most essential step in image processing. Given an input image, it processes the image, extracts essential features, and provides an output image with extraneous and redundant data removed. As a result of this, the memory required to store image data and the

time required to perform further analysis on the image are drastically reduced. The following preprocessing steps are carried out on our input images.

1. Contrast Normalization - Improving the contrast in dark images.
2. Noise Removal - Removal of noise from images using the Non-local Means Denoising algorithm.
3. Binarization - Converting image into binary format.
4. Thinning - Producing skeletonized image (single pixel width).
5. Sampling - Removing additional padding around the foreground pixels in the given image.

A. Support Vector Machine

Support Vector Machine is a supervised learning model. Given a set of training data along with the labels for each category, the SVM algorithm builds a model and outputs optimal hyperplanes that separates categories from each other. This model can later be used to classify new data. Though SVMs are basically used for binary classification, they can be extended to perform multiclass classification. OpenCV is used to implement SVM model. The feature set taken into consideration is Histogram of Oriented Gradients (HOG).

B. K - Nearest Neighbors

k-NN or k-nearest neighbors algorithm is used to classify based on a majority vote of its k neighbors. k is a positive integer, and its value is $\sqrt{n}$ where n is the number of samples. K-NN uses a lazy learning approach where computation is done at the time of classification. No explicit training step is required and it is very sensitive to the local structure of the data. Euclidean Distance is used as a distance metric. It is calculated using,

$$d_E(x, y) = \sum_{i=1}^{N} \sqrt{x^2{}_i - y^2{}_i}$$

Intensity is used as a feature set and the images are stored as numpy arrays for classification. OpenCV's k-NN classifier is run on the dataset with train and test labels. For classification, the test data is appended to the dataset and the majority of the nearest neighbors the image corresponds to is given as the classification. An optimal value of k depends on the data. Larger values of k reduces noise but thins the boundaries of classification. Here, 100 images are used per character. Hence k is best chosen to be square root of 100, or 10.

C. Convolutional Neural Network

Convolutional networks also known as Convolutional Neural Networks, or CNNs, are a specialized kind of neural network for processing data that have a known grid-like topology. Keras [17] is a deep learning library that provides a simplified approach to creating CNNs. It uses Tensorflow as a backend. It is built by using two stacks of Convolution layers and Max Pooling layers. After the first stack, a Local Response Normalization layer is added to implement lateral inhibition. A Dropout layer is also added with a probability of 0.5 to avoid

overfitting. A Flattening layer is then added to convert the feature set to a 1- Dimensional vector which is in turn fed to our fully connected Dense layer. A Softmax function is used as the output layer to get the outputs as probabilities. The first convolution layer uses 32 filters with window size 5x5 and the second convolution layer uses 64 filters of the same window size. Both the convolution layers use sigmoid as the activation function.

D. Transfer Learning

Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.
The benefits of Transfer Learning are that it can speed up the time it takes to develop and train a model by reusing these pieces or modules of already developed models. This helps speed up the model training process and accelerate results. The benefits of Transfer Learning are that it can speed up the time it takes to develop and train a model by reusing these pieces or modules of already developed models. This helps speed up the model training process and accelerate results.

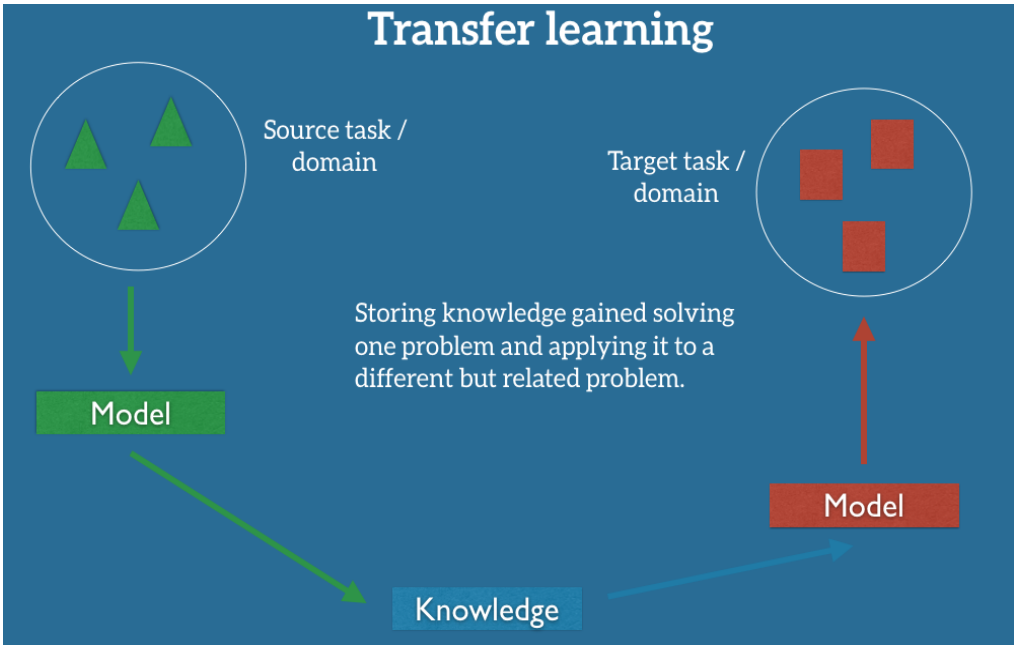

Fig.2. Transfer learning

E. Tensorflow

TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning (also called neural networking) models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++.
TensorFlow allows developers to create *dataflow graphs*—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a

mathematical operation, and each connection or edge between nodes is a multidimensional data array, or *tensor*.

F. Keras

Keras was created to be user friendly, modular, easy to extend, and to work with Python. The API was "designed for human beings, not machines," and "follows best practices for reducing cognitive load." The biggest reasons to use Keras stem from its guiding principles, primarily the one about being user friendly. Beyond ease of learning and ease of model building, Keras offers the advantages of broad adoption, support for a wide range of production deployment options, integration with at least five back-end engines (TensorFlow, CNTK, Theano, MXNet, and PlaidML), and strong support for multiple GPUs and distributed training. Plus, Keras is backed by Google, Microsoft, Amazon, Apple, Nvidia, Uber, and others.

G. Conv2D Layer

Keras Conv2D is a 2D Convolution Layer, this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.
Kernel: In image processing kernel is a convolution matrix or masks which can be used for blurring, sharpening, embossing, edge detection, and more by doing a convolution between a kernel and an image.
The Keras Conv2D class constructor has the following arguments:

```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

Fig.3. Conv2D Layer

Arguments:

- filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- kernel_size: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- strides: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.
- padding: one of "valid" or "same" (case-insensitive). "valid" means no padding. "same" results in padding evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input.
- data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch_size, height, width, channels) while channels_first corresponds to inputs with shape (batch_size, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be channels_last.
- dilation_rate: an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.
- groups: A positive integer specifying the number of groups in which the input is split along the channel axis. Each group is convolved separately with filters / groups filters. The output is the concatenation of all the groups results along the channel axis. Input channels and filters must both be divisible by groups.
- activation: Activation function to use. If you don't specify anything, no activation is applied (see keras.activations).
- use_bias: Boolean, whether the layer uses a bias vector.
- kernel_initializer: Initializer for the kernel weights matrix (see keras.initializers).
- bias_initializer: Initializer for the bias vector (see keras.initializers).
- kernel_regularizer: Regularizer function applied to the kernel weights matrix (see keras.regularizers).
- bias_regularizer: Regularizer function applied to the bias vector (see keras.regularizers).
- activity_regularizer: Regularizer function applied to the output of the layer (its "activation") (see keras.regularizers).
- kernel_constraint: Constraint function applied to the kernel matrix (see keras.constraints).
- bias_constraint: Constraint function applied to the bias vector (see keras.constraints).

H. Flatten Layer

```
tf.keras.layers.Flatten(data_format=None, **kwargs)
```

Fig.4. Flatten Layer

Flattens the input. Does not affect the batch size.
Arguments:

- data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, ..., channels) while channels_first corresponds to inputs with shape (batch, channels, ...). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

I. Inception V3

This model works on the concept of Transfer Learning. The inception model is a convolutional net created by Google as part of LeNet project to extract classes of objects from images. The Inception architecture of GoogLeNet was also designed to perform well even under strict constraints on memory and computational budget. The computational cost of Inception is also much lower than VGGNet and its higher performing successors [20]. The image goes through an image conversion layer which consists of 2 sublayers. They are
Convolution: This layer spans throughout the whole image, using a small filtering kernel of size MxN to come up with a new small image of K layers.

1. Convolution: This layer spans throughout the whole image, using a small filtering kernel of size MxN to come up with a new small image of K layers.
2. Subsampling: Also called as max pooling, takes as input the image generated from convolution layer. It also spans a small stride P x Q throughout the whole image and picks up generally the highest value in that stride and continues to next stride. The last layer is the fully connected layer. This layer takes as input the output from last max pooling layer and is usually a network of further neuron layers and gives as output the softmax layer (prediction of classes).

J. Machine Learning Pipline

A machine learning pipeline is used to help automate machine learning workflows. They operate by enabling a sequence of data to be transformed and correlated together in a model that can be tested and evaluated to achieve an outcome, whether positive or negative.
Machine learning (ML) pipelines consist of several steps to train a model. Machine learning pipelines are iterative as every step is repeated to continuously improve the accuracy of the model and achieve a successful algorithm. To build better machine learning models, and get

the most value from them, accessible, scalable, and durable storage solutions are imperative, paving the way for on-premises object storage.

3. Results

The Chars74K dataset is currently being trained and tested.

4. Scope for future work

1. People with learning disabilities who have difficulty large amount of text due to dyslexia or other problems really benefit if the similar approach is used to develop their model.
2. A similar model can be developed such as Screen readers for people with difficulties reading computer screen.

5. References

1. http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/