# Signboard Translation From Vernacular Languages

**A PROJECT REPORT**

***Submitted by***

**VISHNU TEJA CHIKKALA [RA1711003030378]**
**RAJPREET SRIVASTAV [RA1711003030380]**

*Under the guidance of*
**MR. Sunil Kumar**
(Assistant Professor, Department of Computer Sciene & Engineering)

***in partial fulfillment for the award of the degree of***

**BACHELOR OF TECHNOLOGY**

in

**COMPUTER SCIENCE & ENGINEERING**

of

**FACULTY OF ENGINEERING AND TECHNOLOGY**


SRM INSTITUTE OF SCIENCE & TECHNOLOGY, NCR CAMPUS

**MAY 2021**

# SRM INSTITUTE OF SCIENCE & TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

## BONAFIDE CERTIFICATE

Certified that this project report titled **Signboard Translation from Vernacular Languages** is the bonafide work of **VISHNU TEJA CHIKKALA [RA1711003030378], RAJPREET SRIVASTAV [RA1711003030383]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project re- port or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

**SIGNATURE**

MR.SUNIL KUMAR
**GUIDE**
Assistant Professor
Dept. of Computer Science & Engineering

**SIGNATURE**

Dr. R.P.MAHAPATRA
**HEAD OF THE DEPARTMENT**
Dept. of Computer Science & Engineering

Signature of the Internal Examiner

Signature of the External Examiner

# ABSTRACT

India has 22 constitutionally recognized languages written in 13 different scripts. An average traveler, on a business trip, when travelling to a new region, might often get confused with signboards written in an unfamiliar language. It is also impossible to have every signboard in every city / town / village written in 22 different languages, as there will not be enough room to accommodate more than 2 – 3 scripts.

The objective of this project is to develop a simple, easy-to-use and scalable Android mobile app which provides a two-click, picture-to-text, translation / transliteration service for Indian vernacular languages, using deep neural networks and natural language processing models trained for text detection, recognition and translation tasks on collected and freely-available datasets, using frameworks such as PyTorch and Tensorflow.

For the scope of this project, we will design a system which works for names (such as road names, city names, shop names, organization names, etc.) which typically are not longer than 4-5 words, and support translation for 1 language. Further scope for the project involves including support for more languages and building models to support translation / transliteration of longer pieces of text

# ACKNOWLEDGEMENT

We would like to express our deepest gratitude to our guide, **MR. SUNIL KUMAR** for his valuable guidance, consistent encouragement, personal caring, timely help and providing us with an excellent atmosphere for doing research. All through the work, in spite of his busy schedule, he has extended cheerful and cordial support to us for completing this research work.

**Authors**

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABBREVIATIONS

| | |
|---|---|
| **ANN** - | Artificial Neural Network |
| **CNN** - | Convulutional Neural Network |
| **RNN** - | Recurrent Neural Network |
| **GRU** - | Gated Recurrent Unit |
| **CTC** - | Connectionist Temporal Classification |
| **NITS-LD** - | NIT Silchar Language Database |
| **OGI-MLTS** - | OGI-Multilingual Database |
| **RRNN** - | Regression Residual Neural Network |
| **NLP** - | Natural Language Processing |
| **COCO** - | Common Objects in Context |
| **FPN** - | Feature Pyramid Network |
| **ROI** - | Regions of Interest |
| **SVM** - | Support Vector Machines |

# CHAPTER 1

# INTRODUCTION

## 1.1 Project Overview

The project describes the process of creating an end-to-end stack of neural network models, to perform the tasks of text detection, text recognition and text transliteration in order to extract text of the source language from an image, and output transliterated text in the target language. For this purpose, three models are built:

- a Region-based Convolutional Neural Network for performing text detection (a subset task of object detection with class 'text')
- a Convolutional Neural Network (CNN) - Recurrent Neural Network (RNN) / Gated Recurrent Unit (GRU) Encoder-Decoder Architecture for performing text recognition
- a GRU - GRU Encoder-Decoder Architecture for performing text transliteration

This models will then be trained on a custom dataset featuring a large number of synthetically generated scene-text images, as well a collection of natural scene-text images, along with annotations describing bounding boxes and ground truth source script. The transliteration model will be trained on another dataset which includes large number of source script words and their corresponding target script words.

This model will then be compiled into an end-to-end stack using TensorflowLite and deployed onto an Android app developed in Java.

# CHAPTER 2

# LITERATURE REVIEW

## 2.1 Summary of Literature Review

C.Kurian et al. concluded that the Indian community faces a “Digital Divide” due to dominance of English as mode of communication in higher education, judiciary, corporate sector and Public administration at Central level whereas the government in states work in their respective regional languages[1].

From the 2011 Census of India, it is observed that. while 99 % of the population speak one of these scheduled languages in various dialects (which number in the thousands), according to Census 2011, the total percentage of English speakers is at 10 %, and that too is skewed towards the urban population [2].

N.P. Desai et al noted that traditionally NLP had been approached with statistical methods such as Hidden Markov Machines (HMM), Support vector machine(SVM), Conditional Random Field(CRF), Naive Bayes(NB), etc, which take a large amount of tagged/annotated data (corpus) to statistically analyze and learn the language characteristics , and suggested that deep learning methods or a ‘connectionist approach’ could return better results[3]. The reasons for the same suggested by them and other authors, in brief, were -

i. the simplicity of the solution in rapidly prototyping and establishing practically effffective systems

ii. the lower cost of annotation of the training data [4],

iii. they attempt to more closely emulate the learning process of biological brains [3] [5] [6], among other reasons.

However, A. Kunchukuttan et al., note that the collection of a uniform corpus and standard datasets for training models remains a challenge across all regional languages. The large number of morphological variations across Indic languages also contributes to this issue[7]. This study also proposed the creation of a “large-scale, general-domain” corpus for 10 Indian Languages across 2 scripts[7]. J. Philip et al., in their study, also highlighted several efforts at collating standard datasets and corpora for Indian languages[4].

Sharma et al., 2017 concluded that almost all existing Indian language machine

transliteration systems are based on statistical and hybrid approach[8].

A few models proposed in this field are as follows-

| **Authors** | **Model** | **Accuracy** |
|---|---|---|
| Bhanja et al., 2019 [9] | CNN-LSTM Architecture using ResNet | 93% (NITS-LD), 89% (OGI-MLTS) |
| J. Philip et al, 2019 [4] | IL-Multi, | 34% (cold-start), 40% (with transferred learning) |
| Arafat, S. Y. et al, 2020 [10] | FasterR-CNN - CNN - RRNN Architecture | 99% |

Table 1 List of Proposed Models

## 2.2 Research Gap

From the above literature review, we conclude that there lies a need for developing NLP architectures for facilitating flow of digital content and information in and between local, national and international levels.

The above also means that a large percentage of the literate population is either monolingual or bilingual, and across 22 languages, an accessible, easy-to-use and intuitively developed system is required, which enables intercommunication.

A research gap that we identified was that, though several successful efforts have been made to develop computer vision and natural language processing (NLP) models for working with Indian languages, the corpora and the model techniques used are scattered and are not easily avaliable for general purpose training and use. Thus, with this project, we aimed to develop a standard, general model framework / architecture that could be trained quickly on multiple languages, given sufficient dataset and corpora resources.

# CHAPTER 3

# METHODOLOGY

This chapter discusses the methods, algorithms and techniques used to achieve the project objective in a serialized manner. In short, these steps, in order, are the preparation of datasets to be used with models, defining a model for text detection, from this detected piece of text performing text recognition to extract the script as a string, then transliterate this script into target language script. These trained models are then deployed as part of the mobile app to run inference on user-captured images.

## 3.1 Model Framework

The model framework is comprised of three models which are stacked end-to-end. This means that the output of the first model is formatted and directly fed as input to the second model and so on. All the models are trained independently for achieving higher training accuracy, while the stack of models for text detection and text recognition are both trained end-to-end as well, for tighter bounding box generation and text extraction..

### 3.1.1 Data Collection and Preprocessing

The training dataset for the text detection and text recognition tasks consists of a large number of images containing scene text,synthetically generated. Each of the images has a corresponding annotation, listing the bounding boxes (in RBOX form) and ground truth text script present in the images.


Figure 1: Synthetically Generated Training Image

15.025299 79.619064 91.971375 27.37761 111.49409 87.36937 120.44259 144.5673 और
195.26416 345.93964 346.07916 195.40369 296.7271 296.54498 411.9508 412.13293 किस
544.8015 579.83813 541.4978 506.46115 42.720642 60.455795 136.19897 118.46382 दिन
275.59427 311.88095 302.1434 265.85672 134.48518 159.5067 173.62825 148.60674 रूप
30.469978 163.88913 164.9093 31.490135 182.98358 181.51782 274.3758 275.84155 इस
33.57235 184.95844 185.26584 33.879738 354.1837 353.62552 436.9943 437.55246 तरह
-2.6164436 45.761616 53.37768 4.9996223 155.51007 137.70114 158.39023 176.19916 साथ
343.8163 512.32336 512.94495 344.4379 134.2903 132.54332 192.49599 194.24297 एकाएक
337.52948 504.81384 505.13098 337.84662 241.95956 240.91986 291.94846 292.98816 रमानाथ
507.9361 555.0499 523.0799 475.96606 4.7673645 56.682793 85.69593 33.780502 गया

Figure 2 Training Annotation

For the text detection model, since we are using a model pretrained on the COCO dataset, we format the annotations of the whole dataframe into a JSON file, converted into the COCO format, specifically the BoxMode.XYWH_ABS format.


```
{
  "image_id": 0,
  "file_name": "401.jpeg",
  "height": 500,
  "width": 715,
  "annotations": [
    {
      "bbox": [
        67,
        200,
        187,
        59
      ],
      "bbox_mode": 1,
      "category_id": "0"
    },
    {
      "bbox": [
        279,
        170,
        417,
        38
      ],
      "bbox_mode": 1,
      "category_id": "0"
    }
  ]
}
```


Figure 3 COCO Format Annotation

We have a set of 100k images and their corresponding images available, collected from open sources. However, for experimental purpose we curate a random batch of 2520 images for training, 255 images for validation and 407 images for testing. The ground truth for

these images as Hindi script, which is our source script,, while our target script is English.

The test dataset for text detection and text recognition tasks consists of actual images containing natural scene text.


Figure 4 Detection / Recognition Test Image

The train and test datasets are both in form of xml files containing serialized pairs of source language script and corresponding target language script. (Hindi to English in this case)

```
<?xml version="1.0" encoding="UTF-8"?><TransliterationCorpus CorpusID = "NEWS2012-Training-EnHi-13937" SourceLang = "
English" TargetLang = "Hindi" CorpusType = "Training" CorpusSize = "13937" CorpusFormat = "UTF8">
<Name ID="1">
<SourceName>RAASAVIHAAREE</SourceName>
<TargetName ID="1">रासविहारी</TargetName>
</Name>
<Name ID="2">
<SourceName>DEOGAN ROAD</SourceName>
<TargetName ID="1">देवगन रोड</TargetName>
</Name>
<Name ID="3">
<SourceName>SHATRUMARDAN</SourceName>
<TargetName ID="1">शत्रुमर्दन</TargetName>
</Name>
<Name ID="4">
<SourceName>MAHIJUBA</SourceName>
<TargetName ID="1">महिजुबा</TargetName>
</Name>
<Name ID="5">
<SourceName>SABINE</SourceName>
<TargetName ID="1">सैबिन</TargetName>
</Name>
<Name ID="6">
<SourceName>BILL COSBY</SourceName>
<TargetName ID="1">बिल कॉस्बी</TargetName>
</Name>
<Name ID="7">
<SourceName>RISHTA KAGAZ KA</SourceName>
<TargetName ID="1">रिश्ता कागज़ का</TargetName>
</Name>
<Name ID="8">
```

Figure 5 Transliteration Dataset

### 3.1.2 Text Detection

For the text detection task, we have decided to go with a pretrained model from the model zoo of the Detectron2 kit, specifically the FasterRCNN model (faster_rcnn_R_50_FPN_3x) model with Feature Pyramid Network (FPN; for tightening bounding boxes). This model has been trained on the COCO Dataset for object detection, but we can fine-tune it by doing further training on our model.

A RCNN is an neural network which is trained for both classification and detection of objects in images. It involves a region proposal stage which divides the images to regions-of-interest (ROI) based on some clustering algorithm (which can be both static aur trainable) and some threshold, then a feature extraction stage in which Convolutional Neural Network layers are used to learn meaningful patters from the ROI, before finally passing the extracted features both to a classifier (traditionally an SVM, but a feedforward neural network with softmax output can also be used), and a regression network, where the bounding boxes of the object (if there is a classified object) are learned as a regression problem.

In the case of this particular model, it uses a ResNet as it's backbone CNN, a Region Proposal Network for proposing ROI, and a Box Head for performing regression task to extract bounding boxes.

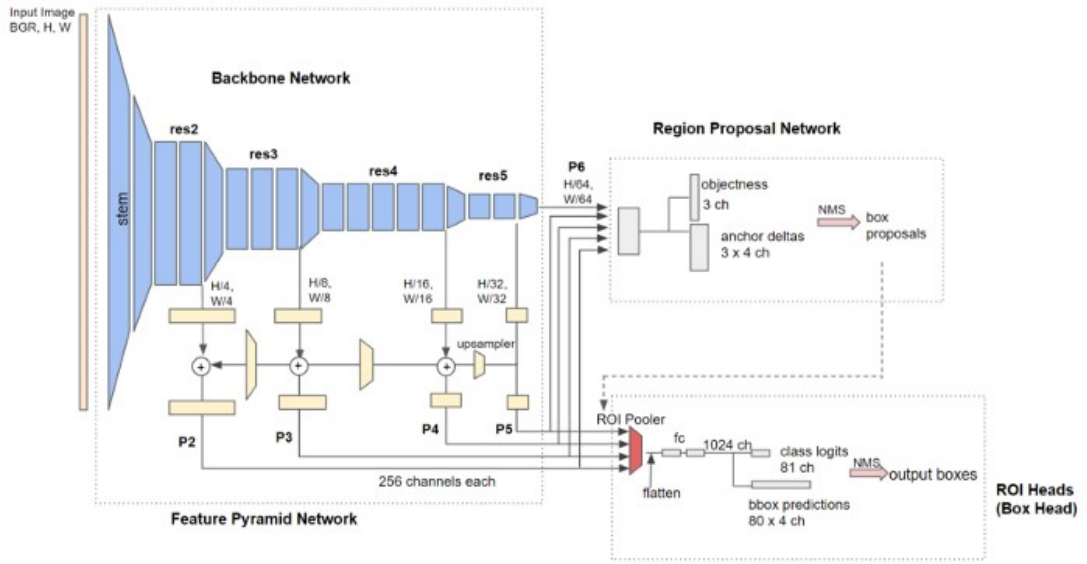



Figure 6 Detectron2 Faster RCNN

Furthermore, since we are only interested in obtaining the bounding boxes of the text from this model, we set the class of all annotations of the training to a single class (in this case, 'Hindi'), and extract the bouding box coordinates as output. The image is then cropped to these bounding box coordinates, and this cropped-bounding-box is fed as input to the next model.

### 3.1.3 Text Recognition

For the text recognition task, we use an encoder-decoder architecture to extract the ground truth script from the cropped bounding box input.

An encoder-decoder architecture is a two-model architecture. The encoder network takes the input to the model and converts it to an intermediate form, while the decoder network then takes this intermediate form as input and produces the target output.

For this model, the encoder is a Convolutional Neural Network (CNN), while the decoder is a Gated Recurrent Unit(GRU). A GRU is a recurrent neural network with information controlling gates, used to deal with variable inputs or inputs that can be represented as a series of timesteps. In this case, a script word is a variable sequence, where every letter in the word can be taken as a timestep. Connectionist Temporal Classifification (CTC) loss will be used to eliminate duplicate recognition of the same letter by adjacent CNN features. Both models are taken from the torch.module packs.

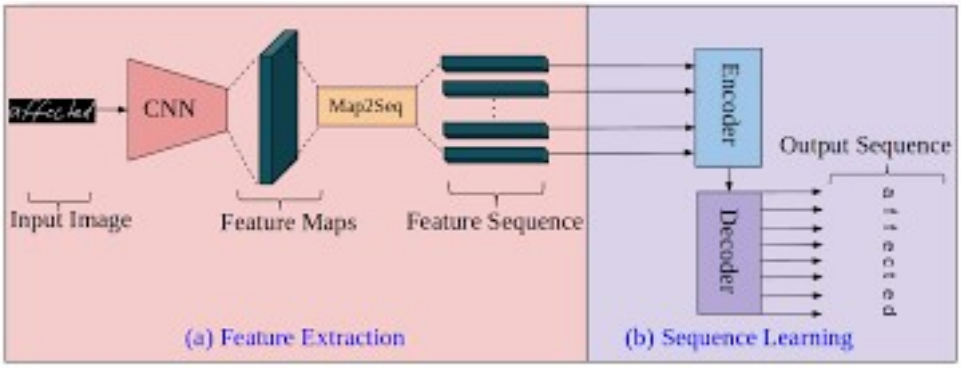



Figure 7 CNN-GRU Text Recognition Model

The text present in the image, in the source language script is output by this model. This string is converted to a one-hot encoded tensor by letter (each character is a one-hot encoded vector, making this output word a tensor of vectors / tensors).. This tensor is then fed to the next and final model as input.

### 3.1.4 Text Transliteration

For the text transliteration task, we use yet another encoder-decoder architecture, but this time both the encoder and decoder models are GRUs, since we are dealing with a sequence-to-sequence generation problem.

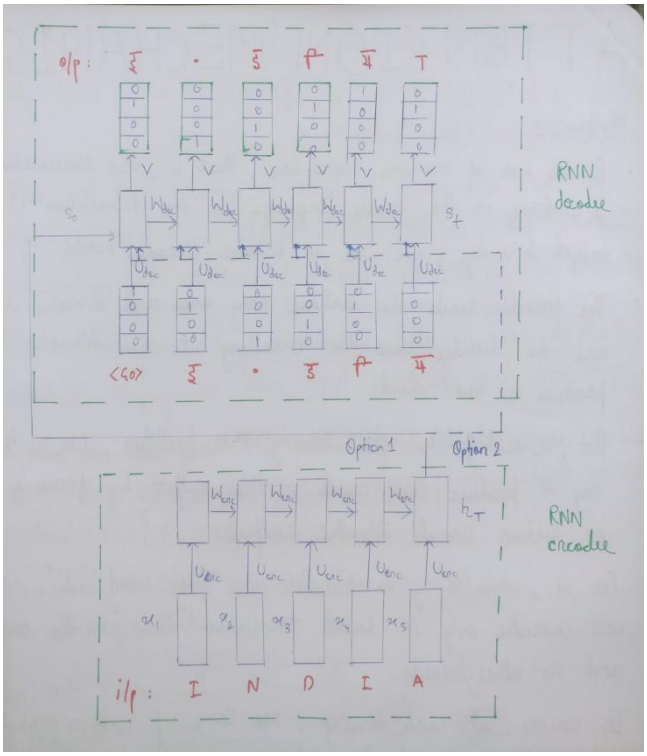


Figure 8 GRU-GRU Text Transliteration Architecture

The source language word, represented as the tensor of tensors, is converted to an intermediate form by the encoder model. From this, the decoder model learns to generate another tensor of one-hot encoded tensors as output. When the one-hot encoding is reversed, we get the output as the corresponding transliteration of source script into target script.

## 3.2 Android App

The models will now be deployed onto an Android app. This app will be developed using Java. The models will be deployed using Tensorflow Lite.

Tensorflow Lite is an open-source framework for on-device inference using models weights saved as tensors. After training, the models weights are checkpointed and saved, then converted into .tflite files by utilizing Tensorflow Lite's Converter, and then deployed onto the app as a package for a function to run inference from.

The app will function as this- the user will click the photo of a signboard, or a road sign, or a shop sign, etc. with some selected source language script. The image will then be captured and sent to the model framework for capturing the text in the image. If some text is detected within a threshold, the model framework will extract and transliterate the text into the target language, and display it to the user.

The app will also use Google Firebase API to provide user functionality and interface.

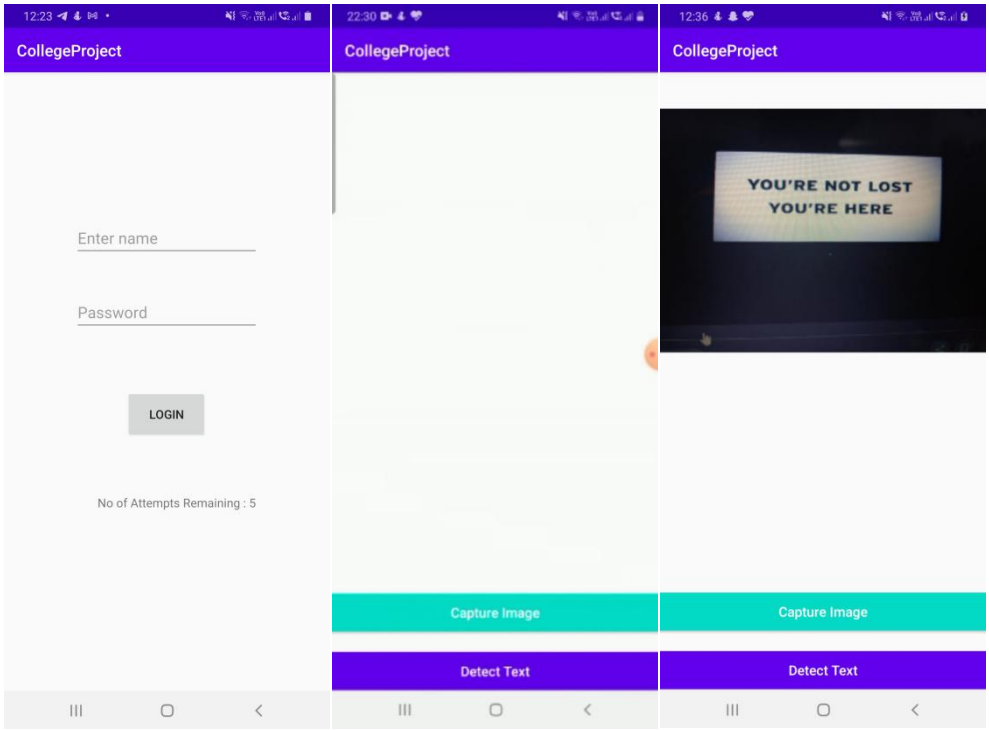



Figure 9: App Interface

# CHAPTER 4

# SYSTEM ARCHITECTURE

The basic functioning of the app is as follows:

1. User captures a photo of the signboard
2. The image is resized so as to be suitable as input to the model
3. The model takes the image as input. The text within the message is detected, extracted and transcribed to target language.
4. The text output (or error message, in case of failure to generate output within threshold confidence), is displayed on screen.

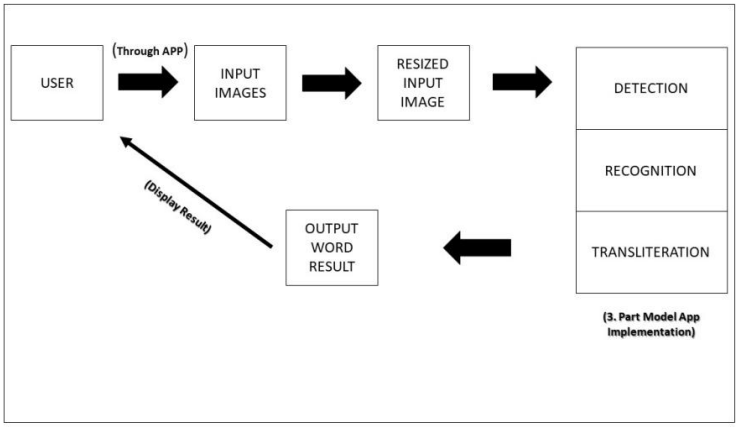



Figure 10: App Architecture

The artifificial neural network (ANN) behind the core functioning of the app is made up of 3 models performing consecutive tasks. That is, the output of a preceding model will be fed as input to the succeeding model, and thus they act as one model unit.

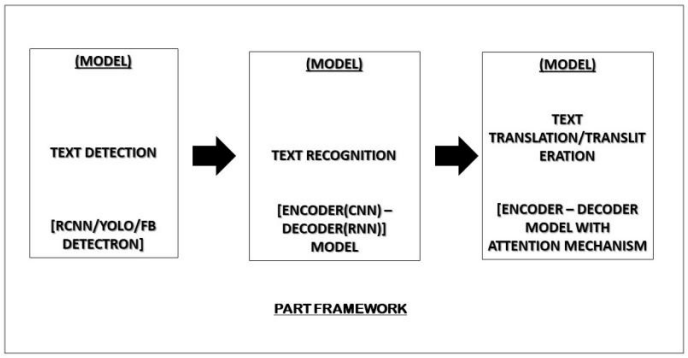



Figure 11: Model Architecture

# CHAPTER 5

# PROJECT CODE AND RESULTS

## MODEL

### Text Detection and Recognition:

```
# install dependencies:
!pip install -U torch==1.5 torchvision==0.6 -f https://download.pytorch.org/whl/cu101/torch_stable.html
!pip install cython pyyaml==5.1
!pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'

import torch, torchvision
print(torch.__version__, torch.cuda.is_available())
!gcc --version

# install detectron2:
!pip install detectron2==0.1.3 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cu101/torch1.5/index.html
```

### Importing Libraries:

```
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# import some common libraries
import numpy as np
import pandas as pd
import cv2
import random
import os
import PIL
from natsort import natsorted
from google.colab.patches import cv2_imshow as gcpimshow

# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog

#import model training libraries
from sklearn.model_selection import train_test_split
```

```
!git clone https://github.com/VishnuTeja-0/vernacular-languages-signboard-translation.git
```

```
Cloning into 'vernacular-languages-signboard-translation'...
remote: warning: multi-pack bitmap is missing required reverse index
remote: Enumerating objects: 235139, done.
remote: Counting objects: 100% (235139/235139), done.
remote: Compressing objects: 100% (234125/234125), done.
remote: Total 235139 (delta 956), reused 235133 (delta 950), pack-reused 0
Receiving objects: 100% (235139/235139), 1.93 GiB | 15.84 MiB/s, done.
Resolving deltas: 100% (956/956), done.
Checking out files: 100% (467254/467254), done.
```

## Dataset Preparation:

```
# Dataset paths
train_images_superfolder_path = "/content/vernacular-languages-signboard-translation/Model/Datasets/Text Detection 2/train/images
train_annotations_superfolder_path = "/content/vernacular-languages-signboard-translation/Model/Datasets/Text Detection 2/train/a

test_images_path = "/content/vernacular-languages-signboard-translation/Model/Datasets/Text Detection 2/dev/images"
test_annotations_path = "/content/vernacular-languages-signboard-translation/Model/Datasets/Text Detection 2/dev/annotations"
```

```
main_result=pd.DataFrame(columns=["file_name","height","width","annotations"])
main_result["annotations"]=main_result["annotations"].astype('object')
```

```
# Checking subfolder iteration
list_of_annotation_files = []
list_of_image_files = []
for annsubdir, imgsubdir in zip(natsorted(os.listdir(train_annotations_superfolder_path)), natsorted(os.listdir(train_images_supe
  annsubdirpath = train_annotations_superfolder_path + "/" + str(annsubdir)
  imgsubdirpath = train_images_superfolder_path + "/" + str(imgsubdir)
  list_of_annotation_files.append(natsorted(os.listdir(annsubdirpath)))
  list_of_image_files.append(natsorted(os.listdir(imgsubdirpath)))
```

```
# converting the above list to dictionaries
image_file_dict = {}
for i in range(len(list_of_image_files)):
  image_file_dict[i+1] = list_of_image_files[i]

annotation_file_dict = {}
for i in range(len(list_of_annotation_files)):
  annotation_file_dict[i+1] = list_of_annotation_files[i]
```

```
# sample of images from dataset
sample_items = random.sample(image_file_dict.items(), 3)
for i in range(3):
  image_loc = train_images_superfolder_path + "/" + str(sample_items[i][0]) + "/" + random.sample(sample_items[0][1], 1)[0]
  img = cv2.imread(image_loc)
  gcpimshow(img)
```

```
word_output_dict = {}
for i in range(1, 26):
  word_output_dict[i] = []
```

```
cat_dict = {"HINDI":"0"}
header_list = ["x1", "x2", "x3", "x4","y1", "y2","y3", "y4", "category_id"]
for key, value in annotation_file_dict.items():
  k = 0
  file_list = value
  for i in file_list:
    file_path = train_annotations_superfolder_path+"/"+str(key)+"/"+i
    df = pd.read_csv(file_path,delimiter = ' ',header = None,index_col=False,names=header_list)
    df["height"] = abs(df["y1"]-df["y3"])
    df["width"] = abs(df["x1"]-df["x3"])
    df=df[["x1","y1","width","height","category_id"]]
    word_output_dict[key].append(df["category_id"])
    # df["category_id"] = "HINDI"
    # df1=df

    # df1["bbox"] = df1.iloc[:,0:4].values.tolist()
    # df1["bbox_mode"] = 1
    # df1 = df1.replace({"category_id": cat_dict})
    # df1=df1[["bbox",→"bbox_mode",——→"category_id"]]
    # annotations = df1.T.to_dict().values()
    # l = []
    # for j in annotations:
    #   l.append(j)
    # res=pd.DataFrame(columns=["file_name","height","width","annotations"])
    # res["annotations"]=res["annotations"].astype('object')
    # res.at[0,"file_name"] = str(key)+"/"+i[:-4]+".jpg"
    # res.at[0,"annotations"] = l
    # h = cv2.imread(train_images_superfolder_path + "/" + str(key) + "/" +str(k)+".jpg").shape[:2]
    # res.at[0,"height"] = h[0]
    # res.at[0,"width"] = h[1]
    # k=k+1
    # main_result = main_result.append(res)
    # main_result.reset_index(drop=True,inplace=True)
```

```
# Splitting dataset into train and validation sets
print(main_result.head)
total_size = main_result.shape
print(total_size)
val_size = total_size[0] // 4
train_size = total_size[0] - val_size
print(total_size, train_size, val_size)
train_main_result = main_result.iloc[0 : train_size]
val_main_result = main_result.iloc[train_size : total_size[0] + 1]
print(train_main_result.shape, val_main_result.shape)
print(train_main_result.head)
print(val_main_result.head)

train_main_result.reset_index(inplace=True)
train_main_result.rename(columns={"index":"image_id"},inplace=True)
train_main_result.to_json("train.json",orient="records")


val_main_result.reset_index(inplace=True)
val_main_result.rename(columns={"index":"image_id"},inplace=True)
val_main_result.to_json("val.json",orient="records")
```

```
# Registering dataset
import json
from detectron2.structures import BoxMode
def get_board_dicts(imgdir, jsontype):
    json_file = jsontype+".json"
    with open(json_file) as f:
        dataset_dicts = json.load(f)
    for i in dataset_dicts:
        filename = i["file_name"]
        i["file_name"] = imgdir + "/" + filename
        for j in i["annotations"]:
            j["bbox_mode"] = BoxMode.XYWH_ABS
    return dataset_dicts

from detectron2.data import DatasetCatalog, MetadataCatalog
for d in ["train", "val"]:
    DatasetCatalog.register("boardetect_" + d, lambda d=d: get_board_dicts(train_images_superfolder_path, d))
    MetadataCatalog.get("boardetect_" + d).set(thing_classes=[0])
```

```
board_metadata = MetadataCatalog.get("boardetect_train")
```

```
print('keep_me_alive')
```

```
keep_me_alive
```

```
# Visualizing training dataset
dataset_dicts = get_board_dicts(train_images_superfolder_path, "train")
for d in random.sample(dataset_dicts, 3):
  img = cv2.imread(d["file_name"])
  visualizer = Visualizer(img[:, :, ::-1], metadata=board_metadata)
  vis = visualizer.draw_dataset_dict(d)
  cv2_imshow(vis.get_image()[:, :, ::-1])
```

## Training – Text Detection

```
from detectron2.engine import DefaultTrainer
from detectron2.evaluation import COCOEvaluator

class CocoTrainer(DefaultTrainer):

  @classmethod
  def build_evaluator(cls, cfg, dataset_name, output_folder=None):

    if output_folder is None:
        os.makedirs("coco_eval", exist_ok=True)
        output_folder = "coco_eval"

    return COCOEvaluator(dataset_name, cfg, False, output_folder)
```

```
# Configuration for training

from detectron2.engine import DefaultTrainer
from detectron2.config import get_cfg
import os
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("boardetect_train",)
cfg.DATASETS.TEST = ("boardetect_val",)
cfg.DATALOADER.NUM_WORKERS = 0
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50_FPN_3x.yaml")  # Let training initialize from m
cfg.SOLVER.IMS_PER_BATCH = 4
cfg.SOLVER.BASE_LR = 0.0125  # pick a good LR
cfg.SOLVER.MAX_ITER = 1500
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 256
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1 # No. of classes = [HINDI]
cfg.TEST.EVAL_PERIOD = 500
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = CocoTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

```
# Training

from detectron2.utils.visualizer import ColorMode

#Use the final weights generated after successful training for inference
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")

cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.8  # set the testing threshold for this model
#Pass the validation dataset
cfg.DATASETS.TEST = ("boardetect_val", )

predictor = DefaultPredictor(cfg)

dataset_dicts = get_board_dicts("Text_Detection_Dataset_COCO_Format/val")
for d in random.sample(dataset_dicts, 3):
    im = cv2.imread(d["file_name"])
    outputs = predictor(im)
    v = Visualizer(im[:, :, ::-1],
                   metadata=board_metadata,
                   scale=0.8,
                   instance_mode=ColorMode.IMAGE
    )
    v = v.draw_instance_predictions(outputs["instances"].to("cpu")) #Passing the predictions to CPU from the GPU
    cv2_imshow(v.get_image()[:, :, ::-1])
```

## MODEL- TEXT RECOGNITION

```
num_chars = len(char2idx)
print(num_chars)
rnn_hidden_size = 256
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
resnet = resnet18(pretrained=True)
```

```
class CRNN(nn.Module):

    def __init__(self, num_chars, rnn_hidden_size=256, dropout=0.1):

        super(CRNN, self).__init__()
        self.num_chars = num_chars
        self.rnn_hidden_size = rnn_hidden_size
        self.dropout = dropout

        # CNN Part 1
        resnet_modules = list(resnet.children())[:-3]
        self.cnn_p1 = nn.Sequential(*resnet_modules)

        # CNN Part 2
        self.cnn_p2 = nn.Sequential(
            nn.Conv2d(256, 256, kernel_size=(3,6), stride=1, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True)
        )
        self.linear1 = nn.Linear(1024, 256)

        # RNN
        self.rnn1 = nn.GRU(input_size=rnn_hidden_size,
                           hidden_size=rnn_hidden_size,
                           bidirectional=True,
                           batch_first=True)
        self.rnn2 = nn.GRU(input_size=rnn_hidden_size,
                           hidden_size=rnn_hidden_size,
                           bidirectional=True,
                           batch_first=True)
        self.linear2 = nn.Linear(self.rnn_hidden_size*2, num_chars)
```

```
def forward(self, batch):

    batch = self.cnn_p1(batch)
    # print(batch.size()) # torch.Size([-1, 256, 4, 13])

    batch = self.cnn_p2(batch) # [batch_size, channels, height, width]
    # print(batch.size())# torch.Size([-1, 256, 4, 10])

    batch = batch.permute(0, 3, 1, 2) # [batch_size, width, channels, height]
    # print(batch.size()) # torch.Size([-1, 10, 256, 4])

    batch_size = batch.size(0)
    T = batch.size(1)
    batch = batch.view(batch_size, T, -1) # [batch_size, T==width, num_features==channels*height]
    # print(batch.size()) # torch.Size([-1, 10, 1024])

    batch = self.linear1(batch)
    # print(batch.size()) # torch.Size([-1, 10, 256])

    batch, hidden = self.rnn1(batch)
    feature_size = batch.size(2)
    batch = batch[:, :, :feature_size//2] + batch[:, :, feature_size//2:]
    # print(batch.size()) # torch.Size([-1, 10, 256])

    batch, hidden = self.rnn2(batch)
    # print(batch.size()) # torch.Size([-1, 10, 512])

    batch = self.linear2(batch)
    # print(batch.size()) # torch.Size([-1, 10, 20])

    batch = batch.permute(1, 0, 2) # [T==10, batch_size, num_classes==num_features]
    # print(batch.size()) # torch.Size([10, -1, 20])

    return batch
```

```
def weights_init(m):
    classname = m.__class__.__name__
    if type(m) in [nn.Linear, nn.Conv2d, nn.Conv1d]:
        torch.nn.init.xavier_uniform_(m.weight)
        if m.bias is not None:
            m.bias.data.fill_(0.01)
    elif classname.find('BatchNorm') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)
```

```
crnn = CRNN(num_chars, rnn_hidden_size=rnn_hidden_size)
crnn.apply(weights_init)
crnn = crnn.to(device)
```

## DATA PREPARATION- TEXT RECOGINITION

```
# Hindi Alphabet Set
# Hindi Unicode Hex Range is 2304:2432. (Source: https://sites.psu.edu/symbolcodes/languages/southasia/devanagari/devanagarichart

hindi_alphabets = [chr(alpha) for alpha in range(2304, 2432)]
hindi_alphabet_size = len(hindi_alphabets)

hindi_alpha2index = {pad_char: 0}
for index, alpha in enumerate(hindi_alphabets):
    hindi_alpha2index[alpha] = index+1

print(hindi_alpha2index)
```

```
def cleanHindiVocab(line):
    line = line.replace('-', ' ').replace(',', ' ')           # Replaces hyphens and commas with spaces from line, so as to get
    # Builds a string by dropping any character which is not in the Devanagari script unicode range, and is not a space
    cleaned_line = ''
    for char in line:
        if char in hindi_alpha2index or char == ' ':
            cleaned_line += char
    return cleaned_line.split()
```

```
# This is for the output word, which is in Hindi
# We shall treat this as a classification task, that is, each letter (one-hot encoded vector) from English input is classified as
# Thus, the output representation is a vector of class labels
def gt_rep(word, letter2index, device = 'cpu'):
    gt_rep = torch.zeros([len(word)+1, 1], dtype=torch.long).to(device)
    for letter_index, letter in enumerate(word):
        pos = letter2index[letter]
        gt_rep[letter_index][0] = pos
    gt_rep[letter_index+1][0] = letter2index[pad_char]
    return gt_rep
```

```
hindi_gt = gt_rep(hindi, hindi_alpha2index)
```

```
# Output set of ground truth script value
output_set = {}
for i in range(1, 26):
  output_set[i] = []
for key, value in word_output_dict.items():
  for word in value:
    new_word = cleanHindiVocab(word)
    output_set[i].append(new_word)
```

## DEFINE LOSS

```
criterion = nn.CTCLoss(blank=0)
```

```
def encode_text_batch(text_batch):

    text_batch_targets_lens = [len(text) for text in text_batch]
    text_batch_targets_lens = torch.IntTensor(text_batch_targets_lens)

    text_batch_concat = "".join(text_batch)
    text_batch_targets = [char2idx[c] for c in text_batch_concat]
    text_batch_targets = torch.IntTensor(text_batch_targets)

    return text_batch_targets, text_batch_targets_lens
```

```
def compute_loss(text_batch, text_batch_logits):
    """
    text_batch: list of strings of length equal to batch size
    text_batch_logits: Tensor of size([T, batch_size, num_classes])
    """
    text_batch_logps = F.log_softmax(text_batch_logits, 2) # [T, batch_size, num_classes]
    text_batch_logps_lens = torch.full(size=(text_batch_logps.size(1),),
                                       fill_value=text_batch_logps.size(0),
                                       dtype=torch.int32).to(device) # [batch_size]
    #print(text_batch_logps.shape)
    #print(text_batch_logps_lens)
    text_batch_targets, text_batch_targets_lens = encode_text_batch(text_batch)
    #print(text_batch_targets)
    #print(text_batch_targets_lens)
    loss = criterion(text_batch_logps, text_batch_targets, text_batch_logps_lens, text_batch_targets_lens)

    return loss
```

```
compute_loss(text_batch, text_batch_logits)
```

## TRAINING- TEXT RECOGNITION

```
num_epochs = 50
lr = 0.001
weight_decay = 1e-3
clip_norm = 5
```

```
optimizer = optim.Adam(crnn.parameters(), lr=lr, weight_decay=weight_decay)
lr_scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, verbose=True, patience=5)
```

```
crnn = CRNN(num_chars, rnn_hidden_size=rnn_hidden_size)
crnn.apply(weights_init)
crnn = crnn.to(device)
```

```
epoch_losses = []
iteration_losses = []
num_updates_epochs = []
for epoch in tqdm(range(1, num_epochs+1)):
    epoch_loss_list = []
    num_updates_epoch = 0
    for image_batch, text_batch in tqdm(train_loader, leave=False):
        optimizer.zero_grad()
        text_batch_logits = crnn(image_batch.to(device))
        loss = compute_loss(text_batch, text_batch_logits)
        iteration_loss = loss.item()

        if np.isnan(iteration_loss) or np.isinf(iteration_loss):
            continue

        num_updates_epoch += 1
        iteration_losses.append(iteration_loss)
        epoch_loss_list.append(iteration_loss)
        loss.backward()
        nn.utils.clip_grad_norm_(crnn.parameters(), clip_norm)
        optimizer.step()
```

```
epoch_loss = np.mean(epoch_loss_list)
print("Epoch:{}    Loss:{}    NumUpdates:{}".format(epoch, epoch_loss, num_updates_epoch))
epoch_losses.append(epoch_loss)
num_updates_epochs.append(num_updates_epoch)
lr_scheduler.step(epoch_loss)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
```

## TEXT TRANSLITERATION

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
import torch.nn.functional as F
import numpy as np

# Instantiates the device to be used as GPU/CPU based on availability
device_gpu = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Visualization tools
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import clear_output

import random
```

## ALPHABETS SETUP

```
# English Alphabet Set
# To enable us to write one hot encoded vectors for individual letters, and thus a vector-of-vectors representations of words, we
# Since transliteration is not case sensitive, we'll not include small letters in our letter-space
eng_alphabets = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
# Since we want our input to be of uniform size, we need to define a 'pad' character to fill out smaller words
pad_char = '-PAD-'

# Assigns each character a numerical value
eng_alpha2index = {pad_char: 0}
for index, alpha in enumerate(eng_alphabets):
    eng_alpha2index[alpha] = index+1

print(eng_alpha2index)
```

```
{'-PAD-': 0, 'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5, 'F': 6, 'G': 7, 'H': 8, 'I': 9, 'J': 10, 'K': 11, 'L': 12, 'M': 13, 'N': 1
4, 'O': 15, 'P': 16, 'Q': 17, 'R': 18, 'S': 19, 'T': 20, 'U': 21, 'V': 22, 'W': 23, 'X': 24, 'Y': 25, 'Z': 26}
```

```
# Hindi Alphabet Set
# Hindi Unicode Hex Range is 2304:2432. (Source: https://sites.psu.edu/symbolcodes/languages/southasia/devanagari/devanagarichart

hindi_alphabets = [chr(alpha) for alpha in range(2304, 2432)]
hindi_alphabet_size = len(hindi_alphabets)

hindi_alpha2index = {pad_char: 0}
for index, alpha in enumerate(hindi_alphabets):
    hindi_alpha2index[alpha] = index+1

print(hindi_alpha2index)
```

```
{'-PAD-': 0, 'ँ': 1, 'ं': 2, 'ः': 3, 'ऄ': 4, 'अ': 5, 'आ': 6, 'इ': 7, 'ई': 8, 'उ': 9, 'ऊ': 10, 'ऋ': 11, 'ऌ': 12, 'ऍ': 13,
'ऎ': 14, 'ए': 15, 'ऐ': 16, 'ऑ': 17, 'ऒ': 18, 'ओ': 19, 'औ': 20, 'क': 21, 'ख': 22, 'ग': 23, 'घ': 24, 'ङ': 25, 'च':
26, 'छ': 27, 'ज': 28, 'झ': 29, 'ञ': 30, 'ट': 31, 'ठ': 32, 'ड': 33, 'ढ': 34, 'ण': 35, 'त': 36, 'थ': 37, 'द': 38, 'ध': 39,
'न': 40, 'ऩ': 41, 'प': 42, 'फ': 43, 'ब': 44, 'भ': 45, 'म': 46, 'य': 47, 'र': 48, 'ऱ': 49, 'ल': 50, 'ळ': 51, 'ऴ': 52, 'व': 5
3, 'श': 54, 'ष': 55, 'स': 56, 'ह': 57, '़': 58, 'ऽ': 59, 'ा': 60, 'ि': 61, 'ी': 62, 'ु': 63, 'ू': 64, 'ृ': 65,
'ॄ': 66, 'ॅ': 67, 'ॆ': 68, 'े': 69, 'ॉ': 70, 'ॊ': 71, 'ो': 72, 'ौ': 73, '्': 74, 'ॎ': 75, 'ॏ': 76, 'ॐ': 77, '॑': 78, '॒': 79,
'॓': 80, '॔': 81, 'ॕ': 82, 'ॖ':
83, 'ॗ': 84, 'क़': 85, 'ख़': 86, 'ग़': 87, 'ज़': 88, 'ड़': 89, 'ढ़': 90, 'फ़': 91, 'य़': 96, 'ॠ':
97, 'ॡ': 98, 'ॢ': 99, 'ॣ': 100, '।': 101, '॥': 102, '०': 103, '१': 104, '२': 105, '३': 106, '४': 107, '५': 108, '६': 109, '७': 1
10, '८': 111, '९': 112, '॰': 113, 'ॱ': 114, 'ॲ': 115, 'ॳ': 116, 'ॴ': 117, 'ॵ': 118, 'ॶ': 119, 'ॷ': 120, 'ॸ': 121, 'ॹ': 122,
'ॺ': 123, 'ॻ': 124, 'ॼ': 125, 'ॽ': 126, 'ॾ': 127, 'ॿ': 128}
```

# HELPER FUNCTIONS FOR DATA PRE-PROCESSING

```
import re
non_eng_letters_regex = re.compile('[^a-zA-Z ]')    # Regex which returns all characters which are not small or capital letters,

# Function to remove all English non-letters from a line
def cleanEnglishVocab(line):
    line = line.replace('-', ' ').replace(',', ' ').upper()    # Replaces hyphens and commas with spaces from line, so as to get
    line = non_eng_letters_regex.sub('', line)                 # Applies above regex on line, replacing every non-letter characte
    return line.split()                                        # Splits the line by delimiter (space) into a list and returns it

# Function to remove all Hindi non-letters from a line
# Since we do not have defined regex for Hindi, we'll have to do the above process manually
def cleanHindiVocab(line):
    line = line.replace('-', ' ').replace(',', ' ')            # Replaces hyphens and commas with spaces from line, so as to get
    # Builds a string by dropping any character which is not in the Devanagari script unicode range, and is not a space
    cleaned_line = ''
    for char in line:
        if char in hindi_alpha2index or char == ' ':
            cleaned_line += char
    return cleaned_line.split()                                # Splits the line by delimiter (space) into a list and returns it
```

# DATASET LOADING

```python
from torch.utils.data import Dataset          # We import and extend the dataset class from torch, so we can use its functions v
import xml.etree.ElementTree as ET            # Library to work with XML files

# Class to define and instantiate a dataset
class TransliterationDataLoader(Dataset):
    def __init__(self, filename):
        self.eng_words, self.hindi_words = self.readXmlDataset(filename, cleanHindiVocab)    # Initializes the English and Hindi
        self.shuffle_indices = list(range(len(self.eng_words)))                              # Shuffles the indexes around for vc
        random.shuffle(self.shuffle_indices)
        self.shuffle_start_index = 0

    # Function to return length of word
    def __len__(self):
        return len(self.eng_words)

    # Function to return word from language words-list, given index
    def __getitem__(self, idx):
        return self.eng_words[idx], self.hindi_words[idx]
```

```python
    # Function to parse XML file to extract English and Hindi text words
    # An XML document data is hierarchical on purpose to enable easy parsing
    def readXmlDataset(self, filename, lang_vocab_cleaner):
        transliterationCorpus = ET.parse(filename).getroot()    # Parses every root node (every point)
        lang1_words = []                                        # List of English words
        lang2_words = []                                        # List of Hindi words

        for line in transliterationCorpus:
            wordlist1 = cleanEnglishVocab(line[0].text)         # The first tag of a node contains English text
            wordlist2 = lang_vocab_cleaner(line[1].text)        # The second tag of a node contains Hindi text

            # Skip noisy data
            # Any data where the length of the Englsh and Hindi strings doesn't match are outliers (with letters / sounds fused t
            if len(wordlist1) != len(wordlist2):
                print('Skipping: ', line[0].text, ' - ', line[1].text)
                continue

            # Pass the words of each language through respective pre-processing functions, and append the vector representation t
            for word in wordlist1:
                lang1_words.append(word)
            for word in wordlist2:
                lang2_words.append(word)

        return lang1_words, lang2_words                         # Returns list for both languages
```

```
    # Gets a sample random word from words-list by using the '__getitem__()' function
    def get_random_sample(self):
        return self.__getitem__(np.random.randint(len(self.eng_words)))

    # Starts from a random index and loops over the whole array to gather 'batch_size' number of datapoints
    def get_batch_from_array(self, batch_size, array):
        end = self.shuffle_start_index + batch_size
        batch = []
        # If the batch size from the chosen index exceeds the length of the array, we load the rest of the back by looping back t
        if end >= len(self.eng_words):
            batch = [array[i] for i in self.shuffle_indices[0:end%len(self.eng_words)]]
            end = len(self.eng_words)
        return batch + [array[i] for i in self.shuffle_indices[self.shuffle_start_index : end]]

    # Loads a data batch
    def get_batch(self, batch_size, postprocess = True):
        eng_batch = self.get_batch_from_array(batch_size, self.eng_words)       # Gets a random batch of some size for English wo
        hindi_batch = self.get_batch_from_array(batch_size, self.hindi_words)   # Gets a random batch of some size for Hindi word
        self.shuffle_start_index += batch_size + 1

        # Reshuffle if 1 epoch is complete
        if self.shuffle_start_index >= len(self.eng_words):
            random.shuffle(self.shuffle_indices)
            self.shuffle_start_index = 0

        return eng_batch, hindi_batch
```

```
# Applying above dataloader class onto uploaded XML files
# Training set has 13937 datapoints while Test set has 1000 datapoints
train_data = TransliterationDataLoader('/content/NEWS2012-Training-EnHi-13937.xml')
test_data = TransliterationDataLoader('/content/NEWS2012-Ref-EnHi-1000.xml')
```

```
# Basic Data Visualization
print("Train Set Size:\t", len(train_data))
print("Test Set Size:\t", len(test_data))

print('\nSample data from train-set:')
for i in range(10):
    eng, hindi = train_data.get_random_sample()
    print(eng + ' - ' + hindi)
```

```
Train Set Size:  20641
Test Set Size:   1000

Sample data from train-set:
OMKAR - ओंकार
SADI - सदी
RATNAMALA - रत्नमाला
CHITNIS - चिटनीस
FAMILY - फ़ैमिली
GHOSUNDA - घोसुंदा
MEDAL - मेडल
HENRY - हेनरी
GURUDWARA - गुरूद्वारा
RAINA - रायना
```

## ENCODING WORDS

```python
# This is for the input word, which is in English
# Each word will be represented as a vector of one-hot encoded vectors
def word_rep(word, letter2index, device = 'cpu'):
    rep = torch.zeros(len(word)+1, 1, len(letter2index)).to(device)
    for letter_index, letter in enumerate(word):
        pos = letter2index[letter]
        rep[letter_index][0][pos] = 1
    pad_pos = letter2index[pad_char]
    rep[letter_index+1][0][pad_pos] = 1
    return rep

# This is for the output word, which is in Hindi
# We shall treat this as a classification task, that is, each letter (one-hot encoded vector) from English input is classified as
# Thus, the output representation is a vector of class labels
def gt_rep(word, letter2index, device = 'cpu'):
    gt_rep = torch.zeros([len(word)+1, 1], dtype=torch.long).to(device)
    for letter_index, letter in enumerate(word):
        pos = letter2index[letter]
        gt_rep[letter_index][0] = pos
    gt_rep[letter_index+1][0] = letter2index[pad_char]
    return gt_rep
```

```
eng, hindi = train_data.get_random_sample()
eng_rep = word_rep(eng, eng_alpha2index)
print(eng, eng_rep)
```

```
CATHY tensor([[[0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]],

       [[0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]],

       [[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]],

       [[1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]]])
```

```
hindi_gt = gt_rep(hindi, hindi_alpha2index)
print(hindi, hindi_gt)
```

```
कॅथे tensor([[22],
        [70],
        [38],
        [72],
        [ 0]])
```

NETWORK ARCHITECTURE

```
# Encoder Decoder Model using GRUs with Attention Mechanism
MAX_OUTPUT_CHARS = 30
class Transliteration_EncoderDecoder_Attention(nn.Module):

    def __init__(self, input_size, hidden_size, output_size, verbose=False):
        super(Transliteration_EncoderDecoder_Attention, self).__init__()

        self.hidden_size = hidden_size
        self.output_size = output_size

        self.encoder_rnn_cell = nn.GRU(input_size, hidden_size)
        # Size is increased, as hidden layer at decoder timestep will be augmented with the context vector
        self.decoder_rnn_cell = nn.GRU(hidden_size*2, hidden_size)

        self.h2o = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=2)

        # Parameters for attention mechanism

        # Weight of encoder state vector from current timestep  (whose probability in the distribution is being computed)
        self.U = nn.Linear(self.hidden_size, self.hidden_size)
        # Weight of decoder state vector from previous timestamp
        self.W = nn.Linear(self.hidden_size, self.hidden_size)
        # Weight applied to the non-linearity applied to attention function (non-linearity applied to above two quantities)
        self.attn = nn.Linear(self.hidden_size, 1)
        # Linear layer converting decoder output at timestamp to size of hidden vector (since we will concatenate context vector
        self.out2hidden = nn.Linear(self.output_size, self.hidden_size)

        self.verbose = verbose
```

```
    def forward(self, input, max_output_chars = MAX_OUTPUT_CHARS, device = 'cpu', ground_truth = None):

        # encoder
        encoder_outputs, hidden = self.encoder_rnn_cell(input)
        # We resize the encoder output to be used in computing context vector
        encoder_outputs = encoder_outputs.view(-1, self.hidden_size)

        if self.verbose:
            print('Encoder output', encoder_outputs.shape)

        # decoder
        decoder_state = hidden
        decoder_input = torch.zeros(1, 1, self.output_size).to(device)

        outputs = []
        # Applying coefficient to encoder output
        U = self.U(encoder_outputs)
```

```
        if self.verbose:
            print('Decoder state', decoder_state.shape)
            print('Decoder intermediate input', decoder_input.shape)
            print('U * Encoder output', U.shape)

        # Generating characters timestep by timestep
        for i in range(max_output_chars):

            # Applying coefficient to decoder state, and resizing for addition with encoder output
            W = self.W(decoder_state.view(1, -1).repeat(encoder_outputs.shape[0], 1))
            # Applying coefficient over the attention function, which in turn is a non-linearity over sum of encoder and decoder
            V = self.attn(torch.tanh(U + W))
            # Applying softmax function over V to get a probability distribution
            attn_weights = F.softmax(V.view(1, -1), dim = 1)

            # Printing out the dimensions of the attention parameters
            if self.verbose:
                print('W * Decoder state', W.shape)
                print('V', V.shape)
                print('Attn', attn_weights.shape)

            # Applying attention probability distribution as weights to the respective encoder outputs
            attn_applied = torch.bmm(attn_weights.unsqueeze(0), encoder_outputs.unsqueeze(0))
            # Applying dimension transformation on decoder output from previous timestamp (to be passed as input to current times
            # This allows us to concatenate it with the context vector obtained from the attention mechanism
            embedding = self.out2hidden(decoder_input)
            # Concatenation decoder input and attention output to create the final input it to current timestamp of decoder
            decoder_input = torch.cat((embedding[0], attn_applied[0]), 1).unsqueeze(0)
```

```
            # Printing dimensions of context vector and new decoder input
            if self.verbose:
                print('Attn LC', attn_applied.shape)
                print('Decoder input', decoder_input.shape)

            out, decoder_state = self.decoder_rnn_cell(decoder_input, decoder_state)

            if self.verbose:
                print('Decoder intermediate output', out.shape)

            out = self.h2o(decoder_state)
            out = self.softmax(out)
            outputs.append(out.view(1, -1))

            if self.verbose:
                print('Decoder output', out.shape)
                self.verbose = False

            max_idx = torch.argmax(out, 2, keepdim=True)
            if not ground_truth is None:
                max_idx = ground_truth[i].reshape(1, 1, 1)
            one_hot = torch.zeros(out.shape, device=device)
            one_hot.scatter_(2, max_idx, 1)

            decoder_input = one_hot.detach()

        return outputs
```

```
# Function to draw inference for a single word when passed through model
def infer(net, word, char_limit, device = 'cpu'):
    input = word_rep(word, eng_alpha2index, device)    # Generates numerical representation of input word
    return net(input, char_limit)                      # Passes this representation to model
```

```
net_attn = Transliteration_EncoderDecoder_Attention(len(eng_alpha2index), 256, len(hindi_alpha2index), verbose=True)

# Dimensional inference
out = infer(net_attn, 'INDIA', 30)

# Running inference on above example (input 'INDIA') on untrained model
print(len(out))
for i in range(len(out)):
    print(out[i].shape, list(hindi_alpha2index.keys())[list(hindi_alpha2index.values()).index(torch.argmax(out[i]))])
```

# TRAINING

## CORE TRAINER

```
# Function to execute training of model
def train_batch(net, opt, criterion, batch_size, device = 'cpu', teacher_force = False):

    # Pushing network to device
    net.train().to(device)
    # Setting all gradients to zero
    opt.zero_grad()
    # Getting a randomly generated batch of training input and output
    eng_batch, hindi_batch = train_data.get_batch(batch_size)

    total_loss = 0
    # Iterating through batch
    for i in range(batch_size):

        # Numerical representation of English input
        input = word_rep(eng_batch[i], eng_alpha2index, device)
        # Numerical representation of the Hindi output
        gt = gt_rep(hindi_batch[i], hindi_alpha2index, device)
        # Calling model, with ground truth passed as additional parameter if teacher forcing (will be directly us
        outputs = net(input, gt.shape[0], device, ground_truth = gt if teacher_force else None)

        # Iterating through parameters output by model
        for index, output in enumerate(outputs):
            # Applying loss function
            loss = criterion(output, gt[index]) / batch_size
            # Computing gradients, 'retain_graph' parameter defines whether or not further training will be done
            loss.backward(retain_graph = True)
            # Aggregating loss
            total_loss += loss

    # Applying gradients at the end of batch
    opt.step()
    # Returns average loss across batch
    return total_loss/batch_size
```

## TRAINING HELPER

```
# Function that sets up the hyperparameters for the training and calls the core training function with these hyperparameters
# It also plots the loss per iteration as a graph
def train_setup(net, lr = 0.01, n_batches = 100, batch_size = 10, momentum = 0.9, display_freq=5, device = 'cpu'):

    # Creating network object and moving it to device
    net = net.to(device)
    # Defining loss function (cross-entropy loss here)
    criterion = nn.NLLLoss(ignore_index = -1)
    # Defining optimization function (Adam, here)
    opt = optim.Adam(net.parameters(), lr=lr)
    # Defining the number of batches for which teacher forcing will be carried out (here, it is for 1/3rd of the total batches)
    teacher_force_upto = n_batches//3

    loss_arr = np.zeros(n_batches + 1)
    # Iterating over batches
    for i in range(n_batches):
        # Training function is called with defined hyperparameters, with teacher force value being conditional on batch number
        # Average loss after training on batch i is stored
        loss_arr[i+1] = (loss_arr[i]*i + train_batch(net, opt, criterion, batch_size, device = device, teacher_force = i<teacher_

        # Plots Iteration v Loss graph
        if i%display_freq == display_freq-1:
            clear_output(wait=True)

            print('Iteration', i, 'Loss', loss_arr[i])
            plt.figure()
            plt.plot(loss_arr[1:i], '-*')
            plt.xlabel('Iteration')
            plt.ylabel('Loss')
            plt.show()
            print('\n\n')

    # Checkpoints training for these parameters
    torch.save(net, 'model.pt')
    # Returns the stored losses of every batch as a list
    return loss_arr
```

## TRAINING

```
net = Transliteration_EncoderDecoder_Attention(len(eng_alpha2index), 256, len(hindi_alpha2index))

loss_history = train_setup(net, lr=0.025, n_batches=1500, batch_size = 64, display_freq=100, device = device_gpu)
```

## INFERENCE

```python
# Function which draws inference for a given word using a given model
def test(net, word, device = 'cpu'):
    net = net.eval().to(device)
    outputs = infer(net, word, 30, device)
    hindi_output = ''
    for out in outputs:
        val, indices = out.topk(1)
        index = indices.tolist()[0][0]
        if index == 0:
            break
        hindi_char = hindi_alphabets[index+1]
        hindi_output += hindi_char
    print(word + ' - ' + hindi_output)
    return hindi_output
```

```python
# Function which gives accuracy of model given inference of the above 'test' function (which it calls) on the test data
def calc_accuracy(net, device = 'cpu'):
    net = net.eval().to(device)
    predictions = []
    accuracy = 0
    for i in range(len(test_data)):
        eng, hindi = test_data[i]
        gt = gt_rep(hindi, hindi_alpha2index, device)
        outputs = infer(net, eng, gt.shape[0], device)
        correct = 0
        for index, out in enumerate(outputs):
            val, indices = out.topk(1)
            hindi_pos = indices.tolist()[0]
            if hindi_pos[0] == gt[index][0]:
                correct += 1

        accuracy += correct/gt.shape[0]
    accuracy /= len(test_data)
    return accuracy
```

```python
accuracy = calc_accuracy(net) * 100
print('Accuracy : ', accuracy)
```

## Accuracy Results

Text Detection:

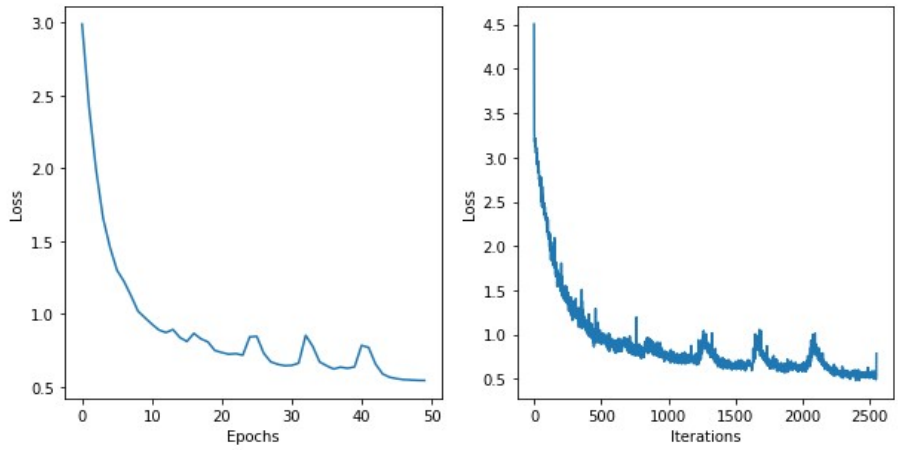
```
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 3 # No. of classes = [HINDI, ENGLISH, OTHER]
cfg.TEST.EVAL_PERIOD = 500
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = CocoTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

```
[08/01 12:28:15 d2.utils.events]:  eta: 0:25:53  iter: 39  total_loss: 1.522  loss_cls: 0.602  loss_box_reg: 0.758  loss_rpn_cls: 0.055  loss_rpn_loc: 0.056  time: 1.0614  data_time
[08/01 12:28:36 d2.utils.events]:  eta: 0:25:06  iter: 59  total_loss: 1.295  loss_cls: 0.395  loss_box_reg: 0.712  loss_rpn_cls: 0.044  loss_rpn_loc: 0.062  time: 1.0534  data_time
[08/01 12:28:57 d2.utils.events]:  eta: 0:24:48  iter: 79  total_loss: 1.110  loss_cls: 0.366  loss_box_reg: 0.632  loss_rpn_cls: 0.026  loss_rpn_loc: 0.051  time: 1.0476  data_time
[08/01 12:29:18 d2.utils.events]:  eta: 0:24:31  iter: 99  total_loss: 1.007  loss_cls: 0.351  loss_box_reg: 0.478  loss_rpn_cls: 0.041  loss_rpn_loc: 0.072  time: 1.0517  data_time
[08/01 12:29:39 d2.utils.events]:  eta: 0:24:14  iter: 119  total_loss: 0.804  loss_cls: 0.251  loss_box_reg: 0.400  loss_rpn_cls: 0.029  loss_rpn_loc: 0.059  time: 1.0510  data_tim
[08/01 12:30:00 d2.utils.events]:  eta: 0:23:48  iter: 139  total_loss: 0.738  loss_cls: 0.230  loss_box_reg: 0.397  loss_rpn_cls: 0.030  loss_rpn_loc: 0.053  time: 1.0483  data_tim
[08/01 12:30:21 d2.utils.events]:  eta: 0:23:28  iter: 159  total_loss: 0.694  loss_cls: 0.229  loss_box_reg: 0.383  loss_rpn_cls: 0.027  loss_rpn_loc: 0.050  time: 1.0494  data_tim
[08/01 12:30:41 d2.utils.events]:  eta: 0:23:03  iter: 179  total_loss: 0.688  loss_cls: 0.224  loss_box_reg: 0.351  loss_rpn_cls: 0.024  loss_rpn_loc: 0.039  time: 1.0466  data_tim
[08/01 12:31:03 d2.utils.events]:  eta: 0:22:45  iter: 199  total_loss: 0.758  loss_cls: 0.247  loss_box_reg: 0.388  loss_rpn_cls: 0.022  loss_rpn_loc: 0.046  time: 1.0475  data_tim
[08/01 12:31:23 d2.utils.events]:  eta: 0:22:21  iter: 219  total_loss: 0.631  loss_cls: 0.221  loss_box_reg: 0.350  loss_rpn_cls: 0.015  loss_rpn_loc: 0.043  time: 1.0467  data_tim
[08/01 12:31:44 d2.utils.events]:  eta: 0:22:00  iter: 239  total_loss: 0.654  loss_cls: 0.200  loss_box_reg: 0.346  loss_rpn_cls: 0.017  loss_rpn_loc: 0.044  time: 1.0454  data_tim
[08/01 12:32:05 d2.utils.events]:  eta: 0:21:38  iter: 259  total_loss: 0.581  loss_cls: 0.194  loss_box_reg: 0.340  loss_rpn_cls: 0.017  loss_rpn_loc: 0.044  time: 1.0444  data_tim
[08/01 12:32:25 d2.utils.events]:  eta: 0:21:20  iter: 279  total_loss: 0.613  loss_cls: 0.184  loss_box_reg: 0.355  loss_rpn_cls: 0.016  loss_rpn_loc: 0.041  time: 1.0439  data_tim
[08/01 12:32:46 d2.utils.events]:  eta: 0:20:56  iter: 299  total_loss: 0.680  loss_cls: 0.217  loss_box_reg: 0.359  loss_rpn_cls: 0.025  loss_rpn_loc: 0.057  time: 1.0440  data_tim
[08/01 12:33:08 d2.utils.events]:  eta: 0:20:39  iter: 319  total_loss: 0.599  loss_cls: 0.197  loss_box_reg: 0.363  loss_rpn_cls: 0.015  loss_rpn_loc: 0.043  time: 1.0470  data_tim
[08/01 12:33:29 d2.utils.events]:  eta: 0:20:18  iter: 339  total_loss: 0.551  loss_cls: 0.172  loss_box_reg: 0.324  loss_rpn_cls: 0.010  loss_rpn_loc: 0.035  time: 1.0468  data_tim
[08/01 12:33:50 d2.utils.events]:  eta: 0:19:57  iter: 359  total_loss: 0.543  loss_cls: 0.150  loss_box_reg: 0.309  loss_rpn_cls: 0.010  loss_rpn_loc: 0.050  time: 1.0467  data_tim
[08/01 12:34:12 d2.utils.events]:  eta: 0:19:38  iter: 379  total_loss: 0.565  loss_cls: 0.163  loss_box_reg: 0.295  loss_rpn_cls: 0.024  loss_rpn_loc: 0.037  time: 1.0480  data_tim
[08/01 12:34:31 d2.utils.events]:  eta: 0:19:14  iter: 399  total_loss: 0.620  loss_cls: 0.212  loss_box_reg: 0.326  loss_rpn_cls: 0.016  loss_rpn_loc: 0.036  time: 1.0448  data_tim
[08/01 12:34:53 d2.utils.events]:  eta: 0:18:54  iter: 419  total_loss: 0.574  loss_cls: 0.167  loss_box_reg: 0.323  loss_rpn_cls: 0.017  loss_rpn_loc: 0.043  time: 1.0460  data_tim
[08/01 12:35:14 d2.utils.events]:  eta: 0:18:34  iter: 439  total_loss: 0.561  loss_cls: 0.160  loss_box_reg: 0.301  loss_rpn_cls: 0.013  loss_rpn_loc: 0.040  time: 1.0468  data_tim
[08/01 12:35:34 d2.utils.events]:  eta: 0:18:10  iter: 459  total_loss: 0.543  loss_cls: 0.171  loss_box_reg: 0.332  loss_rpn_cls: 0.010  loss_rpn_loc: 0.034  time: 1.0440  data_tim
[08/01 12:35:54 d2.utils.events]:  eta: 0:17:48  iter: 479  total_loss: 0.532  loss_cls: 0.172  loss_box_reg: 0.329  loss_rpn_cls: 0.009  loss_rpn_loc: 0.038  time: 1.0438  data_tim
```

```
[07/28 12:59:29 d2.data.common]: Serializing 27 elements to byte tensors and concatenating them all ...
[07/28 12:59:29 d2.data.common]: Serialized dataset takes 0.01 MiB
[07/28 12:59:29 d2.evaluation.evaluator]: Start inference on 27 images
[07/28 12:59:33 d2.evaluation.evaluator]: Inference done 11/27. 0.3363 s / img. ETA=0:00:05
[07/28 12:59:39 d2.evaluation.evaluator]: Inference done 27/27. 0.3246 s / img. ETA=0:00:00
[07/28 12:59:39 d2.evaluation.evaluator]: Total inference time: 0:00:07.273351 (0.330607 s / img per device, on 1 devices)
[07/28 12:59:39 d2.evaluation.evaluator]: Total inference pure compute time: 0:00:07 (0.324569 s / img per device, on 1 devices)
[07/28 12:59:39 d2.evaluation.coco_evaluation]: Preparing results for COCO format ...
[07/28 12:59:39 d2.evaluation.coco_evaluation]: Saving results to /output/coco_instances_results.json
[07/28 12:59:39 d2.evaluation.coco_evaluation]: Evaluating predictions ...
Loading and preparing results...
DONE (t=0.00s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.07s).
Accumulating evaluation results...
DONE (t=0.03s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.486
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.794
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.588
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.416
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.541
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.454
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.195
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.536
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.538
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.498
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.610
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.519
[07/28 12:59:39 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 48.614 | 79.410 | 58.818 | 41.596 | 54.075 | 45.368 |
[07/28 12:59:39 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category   | AP     | category   | AP     | category   | AP     |
|:-----------|:-------|:-----------|:-------|:-----------|:-------|
| HINDI      | 57.467 | ENGLISH    | 45.207 | OTHER      | 43.168 |
OrderedDict([('bbox',
              {'AP': 48.614130895033696,
               'AP-ENGLISH': 45.20707317037124,
```

# Text Recognition

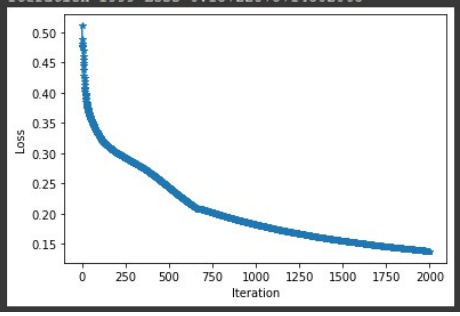
```
ax1.plot(epoch_losses)
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")

ax2.plot(iteration_losses)
ax2.set_xlabel("Iterations")
ax2.set_ylabel("Loss")

plt.show()
```




# Text Transliteration

```
loss_history = train_setup(net_att, lr=0.001, n_batches=2000, batch_size = 64, display_freq=10, device = device_gpu)
```

```
Iteration 1999 Loss 0.13722875714302063
```




```
# Accuracy of the Encoder Decoder model, with and without attention
accuracy = calc_accuracy(net) * 100
accuracy_attn = calc_accuracy(net_att) * 100
print('Accuracy w/o attention ', accuracy)
print('Acurracy with attention', accuracy_attn)
```

```
Accuracy w/o attention  69.25025668775679
Acurracy with attention 69.93080114330115
```

# ANDROID APP

Build Gradle:

```
buildscript {
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:4.0.1"
        classpath 'com.google.gms:google-services:4.3.4'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Dependencies:

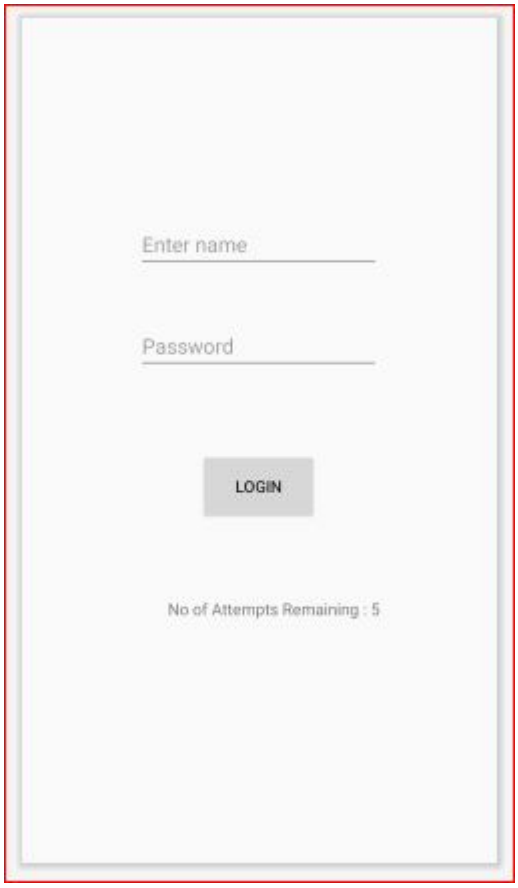
```
dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    implementation 'com.google.firebase:firebase-core:15.0.2'
    implementation 'com.google.firebase:firebase-ml-vision:15.0.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'

}
apply plugin: 'com.google.gms.google-services'
```

**XML Files**:

Main Activity

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/etName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:autofillHints=""
        android:ems="10"
        android:hint="Enter name"
        android:inputType="textPersonName"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.253" />

    <EditText
        android:id="@+id/etPassword"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Password"
        android:inputType="textPassword"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/etName"
        app:layout_constraintVertical_bias="0.092" />

    <Button
        android:id="@+id/btnLogin"
        android:layout_width="102dp"
        android:layout_height="61dp"
        android:text="LOGIN"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/etPassword"
        app:layout_constraintVertical_bias="0.187" />

    <TextView
        android:id="@+id/tvAttemptInfo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=" No of Attempts Remaining : 5"
```

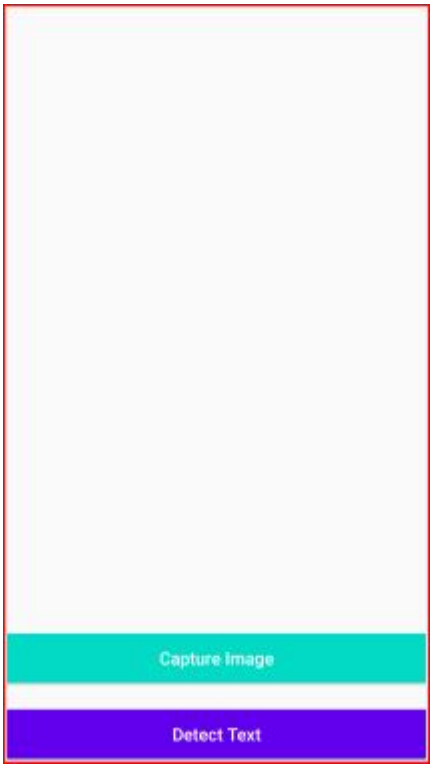
```
        android:text=" No of Attempts Remaining : 5"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.549"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/btnLogin"
        app:layout_constraintVertical_bias="0.238" />

</androidx.constraintlayout.widget.ConstraintLayout>
```




UI FOR LOGIN

1. Text Recognition Screen:

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".TextRecogintion">

    <ImageView
        android:id="@+id/image_view"
        android:layout_width="match_parent"
        android:layout_height="400dp"/>
    <TextView
       android:id="@+id/text_display"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text=""
        android:layout_below="@id/image_view"
        android:textSize="26dp"
        />

    <Button
        android:id="@+id/capture_image"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_above="@+id/detect_text_image_btn"
        android:layout_marginBottom="26dp"
        android:background="@color/colorAccent"
        android:text="Capture Image"
        android:textAllCaps="false"
        android:textColor="@android:color/white"
        android:textSize="17dp" />

    <Button
        android:id="@+id/detect_text_image_btn"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="1dp"
        android:background="@color/colorPrimary"
        android:text="Detect Text"
        android:textAllCaps="false"
        android:textColor="@android:color/white"
        android:textSize="17dp" />


</RelativeLayout>
```




UI FOR CAPTURE IMAGE

3.JAVA Code for Main Login Screen

```
package com.rajpreet.collegeproject;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    private EditText eName;
    private EditText ePassword;
    private Button eLogin;
    private TextView eAttemptsInfo;

    private String Username = "Admin";
    private String Password = "1234";

    boolean isValid = false;

    private int counter = 5;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity main);
```

```
        eName = findViewById(R.id.etName);
        ePassword = findViewById(R.id.etPassword);
        eLogin = findViewById(R.id.btnLogin);
        eAttemptsInfo = findViewById(R.id.tvAttemptInfo);

        eLogin.setOnClickListener((view) -> {

                String inputName = eName.getText().toString();
                String inputPassword = ePassword.getText().toString();

                if(inputName.isEmpty() || inputPassword.isEmpty())
                {
                    Toast.makeText( context: MainActivity.this, text: "Please enter all the details correctly!",Toast.LENGTH_SHORT).show();
                }
                else
                {
                    isValid =  validate(inputName, inputPassword);

                    if (!isValid)
                    {
                        counter--;

                        Toast.makeText( context: MainActivity.this, text: "Incorrect Credentials Entered!",Toast.LENGTH_SHORT).show();

                        eAttemptsInfo.setText( "No of Attempts Remaining : " +counter);

                        if(counter==0){
                            eLogin.setEnabled(false);
                        }

                    }else
                    {
                        Toast.makeText( context: MainActivity.this, text: "Login Successful!",Toast.LENGTH_SHORT).show();

                        Intent intent = new Intent( packageContext: MainActivity.this,TextRecogintion.class);
                        startActivity(intent);
                    }


                }
        });
    }

    private boolean validate(String name, String password){

        if(name.equals(Username) && password.equals(Password))
        {
            return true;
        }

        return false;
    }
}
```

4. Text Recognition Code:

The below code is for the declaration of the buttons and the variables:

```
package com.rajpreet.collegeproject;

import ...

public class TextRecogintion extends AppCompatActivity {

    private Button CaptureImageBtn,detectTextBtn;
    private ImageView imageView;
    private TextView textView;
    static final int REQUEST_IMAGE_CAPTURE = 1;
```

This code is to call the buttons from the xml  document and initialize them :

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_text_recogintion);

        CaptureImageBtn = findViewById(R.id.capture_image);
        detectTextBtn = findViewById(R.id.detect_text_image_btn);
        imageView = findViewById(R.id.image_view);
        textView = findViewById(R.id.text_display);

        CaptureImageBtn.setOnClickListener((view) -> {
                dispatchTakePictureIntent();
                textView.setText("");
        });
        detectTextBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //detectTextFromImage();
            }
        });
    }
```

This code is to take the permission of the user to use camera to capture the image:

```
    private void dispatchTakePictureIntent() {
        Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        try {
            startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
        } catch (ActivityNotFoundException e) {
            // display error state to the user
        }
    }
```

Following code displays the text captured from the image and thus use it as

an input to our model.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {
        Bundle extras = data.getExtras();
        Bitmap imageBitmap = (Bitmap) extras.get("data");
        imageView.setImageBitmap(imageBitmap);
    }
}
```

# CHAPTER 6

# CONCLUSION

The rapid dissemination of infomation in digital format, coupled with the complexities of having such linguistic diversity in our nation, necessitate the development of computer vision and natural language processing models which facilitate communication across languages.

It is also imperative that these solutions be scalable, accessible and distributable to serve the large population of our country.

The aim of the project, to develop a signboard translation system, was an example of how deep learning and artificial neural networks can be harnessed to make life more convenient. It is a simple app which detects, extracts and translates from source to target language, text captured as part of a natural scene by a user, and can be especially useful to interstate and international travellers. Various neural network models were used to train on a set of synthetically generated and curated data and the best performing model, given the time constraints, was selected for evaluation. However, these models, even in this limited scope, still have room for improvement using more diverse testing and use of techniques such as dropout,batch normalization, ensemble modelling, etc.

# CHAPTER 7

# SCOPE FOR FUTURE

The given model is successfully implementated with a modest accuracy, but this project was designed from the get-go to be scalable and improvable.

To start with, the training of the models could have been improved by using techniques such as regularization, dropout, batch normalization and sequence batching to achieve even higher training and testing accuracy. The model could have been further tested on a more diverse range of natural scene images.

This project can also be extended to work with longer sequences of text, and in particular, other languages. Though we have used Hindi and English here as source and target languages, this same model framework can be trained for any combination of source and target languages, given an appropriate curation of dataset. These can then be made available to the user of the app as downloadable packs.

While this project features the raw programming of the models, a long-term view of this project will be to make an easy-to-use API, so that, given the datasets, other developers using natural scene detection, recognition and transliteration as a part of their projects (for example, detection of labels from medicine bottles, or very accurate newsreader apps) can easily train and deploy models.

The app itself can be added to in terms of functionality, and developing a user log-in system, improving UI, history of captured images and their translations, etc. are some features that can be implemented.

# REFERENCES


1. C. Kurian and K. Kannan Balakrishnan.(2008). “Natural language processing in india prospects and challanges”.*In Proceedings of the International Conference on “Recent Trends in Computational Science.*
2. ORGI. "Census of India: Comparative speaker's strength of Scheduled Languages-1951, 1961, 1971, 1981, 1991 ,2001 and 2011"
3. N. P. Desai and V. K. Dabhi. (2021) “Taxonomic survey of hindi language NLP systems”, *arXiv preprint arXiv:2102.00214,*
4. J. Philip, V. P. Namboodiri, and C. Jawahar (2019), “A baseline neural machine translation system for indian languages”, *arXiv preprint arXiv:1907.12437*.
5. M. Rosca and T. Breuel. (2016) ”Sequence-to-sequence neural network models for transliteration.”. *arXiv preprint arXiv:1610.09565*.
6. T. Deselaers, S. Hasan, O. Bender, and H. Ney (2009). “A deep learning approach to machine transliteration”. *In Proceedings of the Fourth Workshop on Statistical Machine Translation, pages 233–241.*
7. A. Kunchukuttan, D. Kakwani, S. Golla, A. Bhattacharyya, M. M. Khapra, P. Kumar, et al. (2020). “Ai4bharat-indicnlp corpus: Monolingual corpora and word embeddings for indic languages.” *arXiv preprint arXiv:2005.00085.*
8. A. Sharma and D. Rattan.(2017) *“*Machine transliteration for indian languages: A review*.” International Journal of Advanced Research in Computer Science, 8(8),*
9. Bhanja, C. C., Bisharad, D., & Laskar, R. H. (2019). “Deep residual networks for pre-classification based Indian language identification*.” Journal of Intelligent & Fuzzy Systems, 36(3), 2207-2218.*
10. Arafat, S. Y., & Iqbal, M. J. (2020). “Urdu-text detection and recognition in natural scene images using deep learning*.” IEEE Access, 8, 96787-96803.*