SimpL Translator

RAHUL 19BAI1121
RISHMA 19BAI1128

Guide
Dr. R. Rajalakshmi

Contents

TOPICS FOR DISCUSSION

Introduction

Machine translation is a domain of computational semantics, which explores the use of software to translate text or speech from one language to another.

Machine translation simply substitutes the words in the input sentence and provides the output.

These word-to-word translations may not assure a good and acceptable translation.

The natural ambiguity and flexibility of human language make it difficult to deduce a perfect method for text translation

Problem Statement

IMPLEMENTATION OF NATURAL LANGUAGE TRANSLATION AND OPTICAL CHARACTER RECOGNITION

Objective

- Training a model to translate text from English to Hindi
- Help real-time communication removing the language barrier
- Extend the project to include image

Intended Users

NMT

The models can further be used in:

- Text Summarization
- Chatbot
- Subject Extraction
- Part of Speech Tagging
- Textual Entailment

Intended Users

OCR

The models can further be used in fields like :

- Banking
- Legal
- Travelling
- Food and Beverages
- Healthcare

Dataset Description

source

tides	39%
ted	31%
Other (37726)	30%

Valid	128k	100%
Mismatched	0	0%
Missing	0	0%
Unique	3	
Most Common	tides	39%

english_sentence

124318
unique values

Valid	128k	100%
Mismatched	0	0%
Missing	2	0%
Unique	124k	
Most Common	(Laughter)	0%

hindi_sentence

97662
unique values

Valid	128k	100%
Mismatched	0	0%
Missing	0	0%
Unique	97.7k	
Most Common	(हँसी)	0%

source	english_se...	hindi_sent...
ted	politicians do not have permission to do what needs to be done.	राजनीतिज्ञों के पास जो कार्य करना चाहिए, वह करने कि अनुमति नहीं है .
ted	I'd like to tell you about one such child,	मई आपको ऐसे ही एक बच्चे के बारे में बताना चाहूंगी,
indic2012	This percentage is even greater than the percentage in India.	यह प्रतिशत भारत में हिन्दुओं प्रतिशत से अधिक है।
ted	what we really mean is that they're bad at not paying attention.	हम ये नहीं कहना चाहते कि वो ध्यान नहीं दे पाते
indic2012	.The ending portion of these Vedas is called Upanishad.	इन्हीं वेदों का अंतिम भाग उपनिषद कहलाता है।

Dataset Description

Total No. of Sentences	Total No of Words	Average word per Sentence	Vocabulary Size
E: 127605 H: 127605	E: 1908266 H: 2399762	E: 14 H: 18	E: 70502 H: 81198

Tools

- JUPYTER NOTEBOOK | GOOGLE COLAB
- PYTHON LIBRARIES

Modules

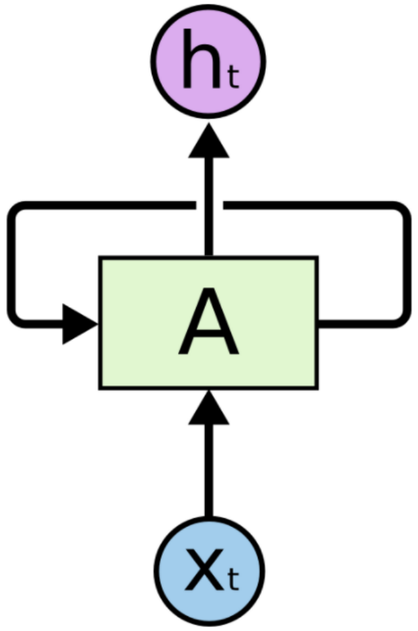
- TEXT TRANSLATION
  - SEQ-SEQ MULTILAYER LSTM
  - SEQ-SEQ BIDIRECTIONAL LSTM
  - ATTENTION MODEL
- OCR
  - ENGLISH
  - TAMIL

Base Model

- Recurrent Neural Network

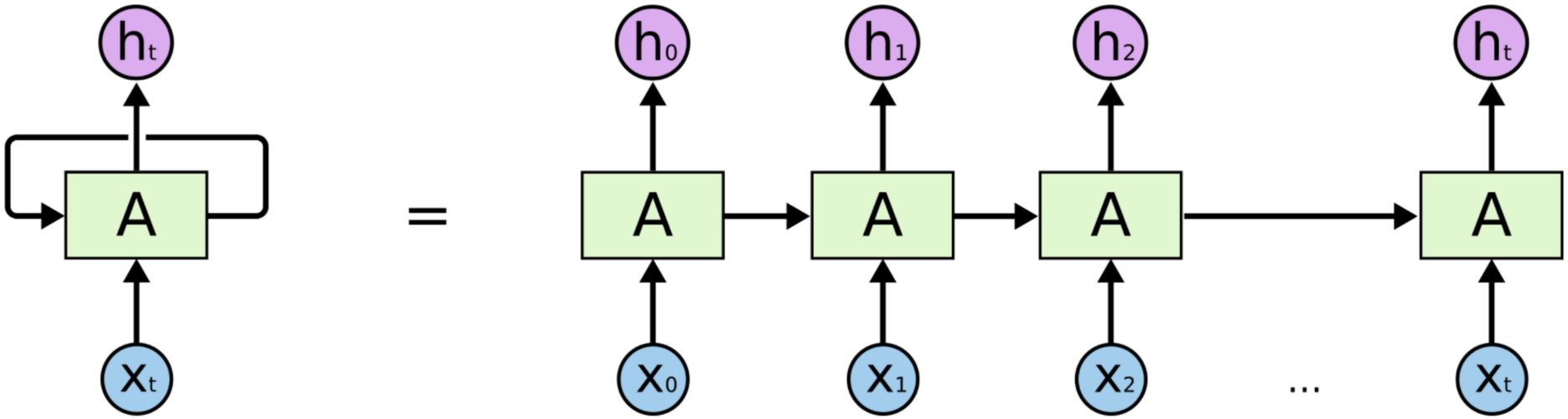
RNN are a type of Neural Network where the output from previous step are fed as input to the current step.

An RNN remembers each and every information through time.This is called Long Short Term Memory.



TEXT TRANSLATION

RNN



BASE MODEL

Base Model

- LSTM: Long Short Term Memory

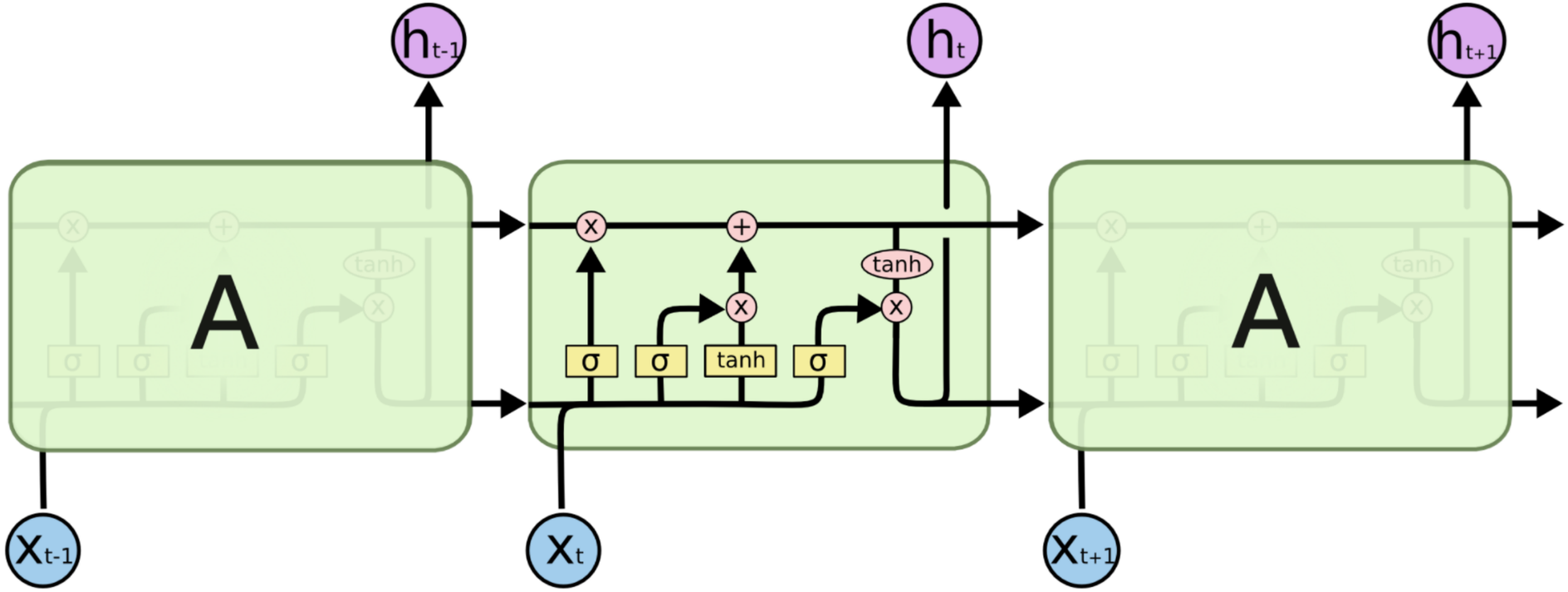
LSTM have a chain like structure containing 4 interacting layers

LSTM contains the cell state where information can flow along it unchanged. We may add or remove data from the cell state using gates

LSTM has three of these gates, to protect and control the cell state: Sigmoid Gates and tanh Gate

TEXT TRANSLATION

LSTM



BASE MODEL

Base Model

- Tokenization

Tokenization breaks raw text into words.

Helps interpret the meaning by analyzing the sequence of the words

Addition of 2 tokens ‘START_’ and ‘_END’ to the Hindi sentences

- Word Embedding

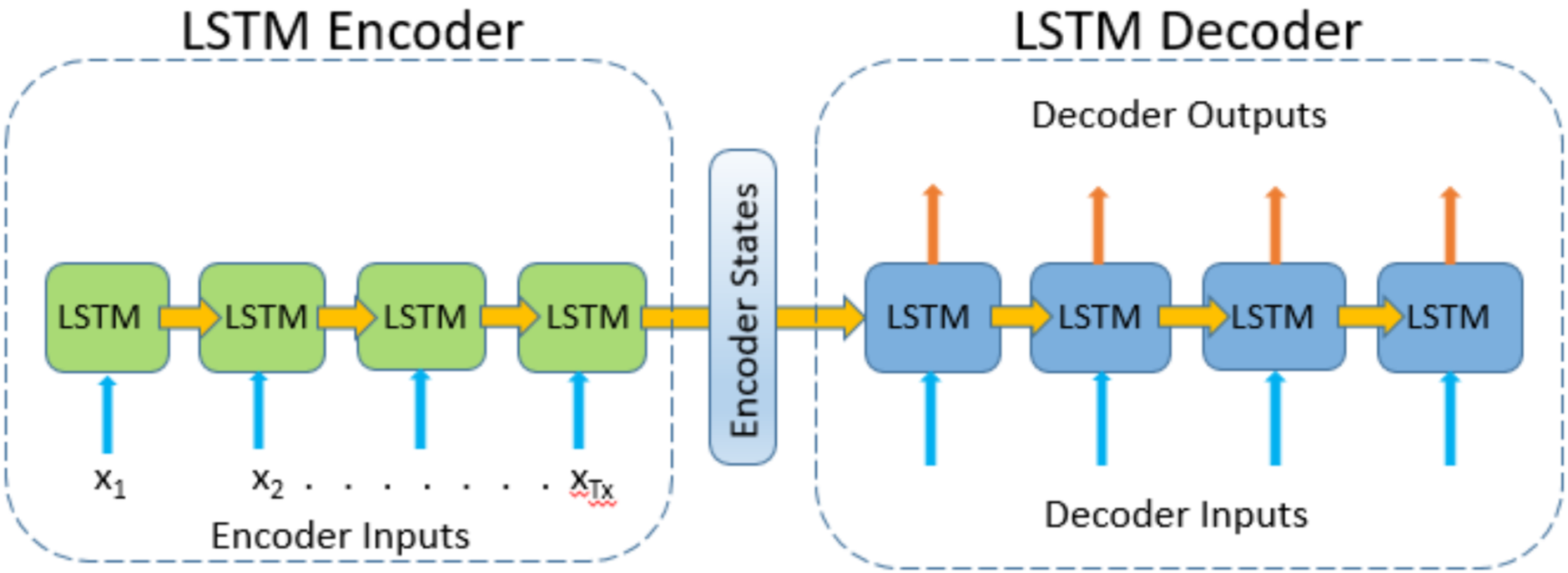
Type of word representation allowing words with similar meaning to have a similar representation

TEXT TRANSLATION

TEXT TRANSLATION

Seq-Seq Multilayer LSTM

Encoder-Decoder Architecture



SEQ-SEQ MULTILAYER LSTM

Encoder-Decoder Architecture

Encoder LSTM

Encoder plays the same role in both training and inference phase

Important components of Encoder LSTM
xi: Input sequence at time step i, represented as vectors using

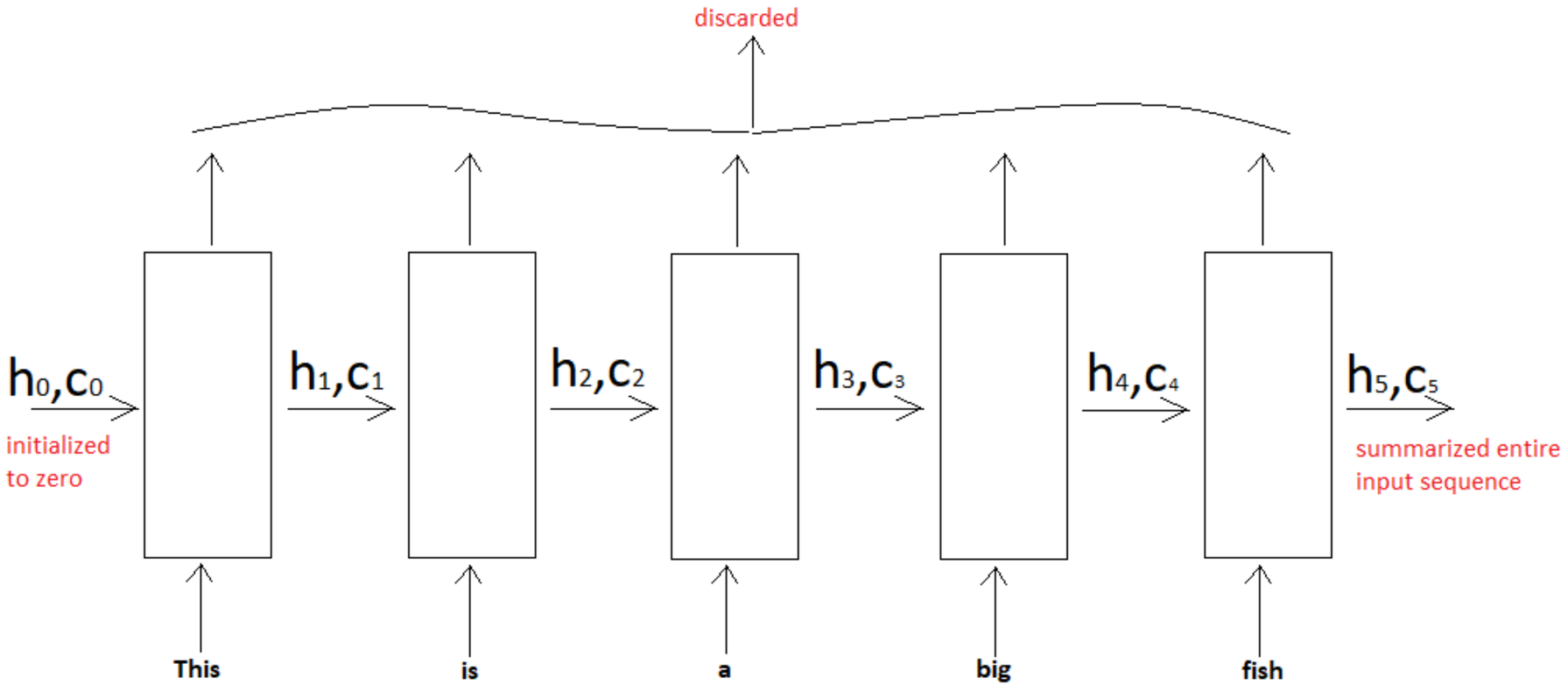
Internal State: hi & ci: Represent what the LSTM has read through each time step i. [Thought Vector]
h0, c0: is initialized to 0
hk, ck: final step known as encoding of xi

yi: Output sequence at step i, discarded in our encoder structure

SEQ-SEQ MULTILAYER LSTM

Encoder-Decoder Architecture

Encoder LSTM



SEQ-SEQ MULTILAYER LSTM

Encoder-Decoder Architecture

Decoder LSTM - Training Mode

Decoder will generate the output sequence word by word

Same Components like Encoder LSTM

START_ - _END: guides the beginning and the ending of a translation

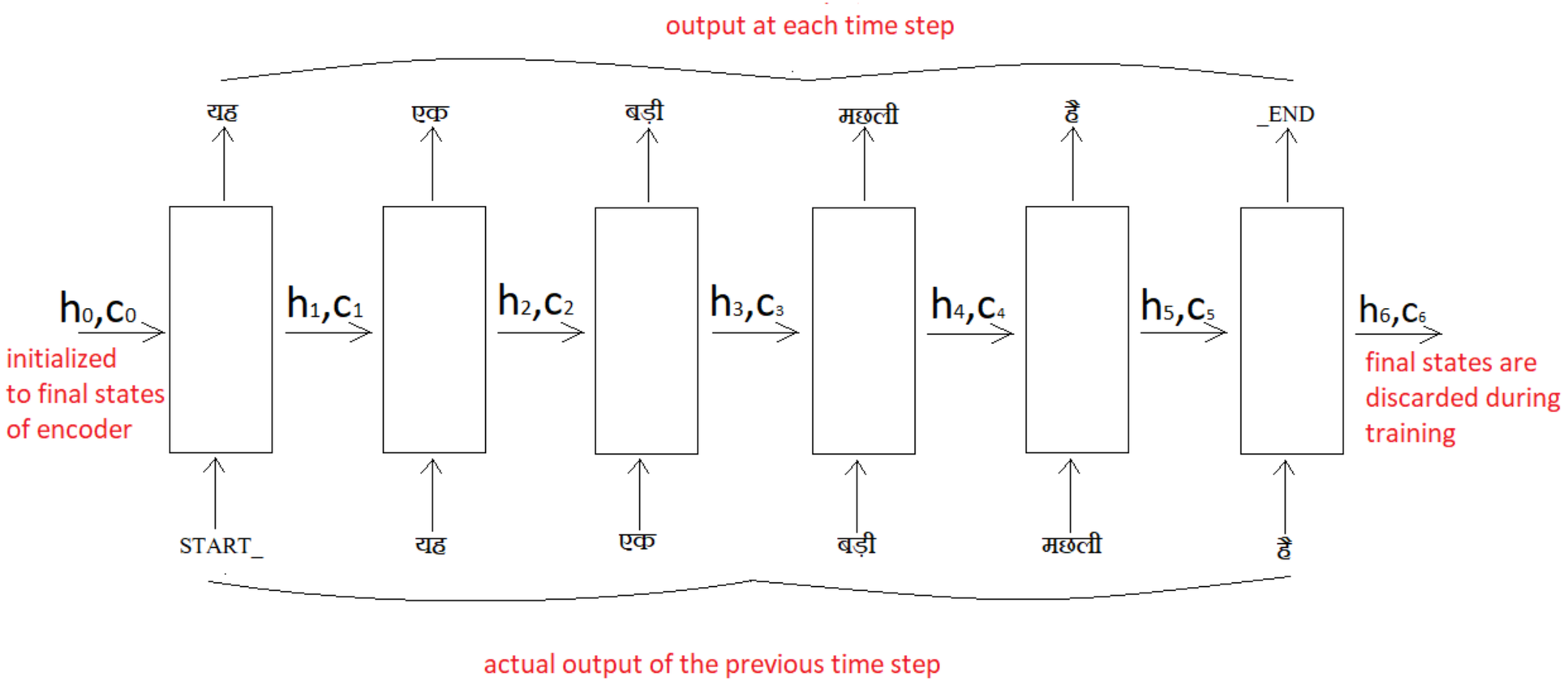
h0, c0: set to the final states of encoder

Teacher Forcing: target word is passed as the next input to the decoder.

SEQ-SEQ MULTILAYER LSTM

Encoder-Decoder Architecture

Decoder LSTM - Training Mode



Encoder-Decoder Architecture

Decoder LSTM - Inference Mode

Decoder will generate the entire output sequence given the thought vectors, word by word.

The initial states of the decoder are set to the final states of the encoder.

The initial input to the decoder is always the START_ token.

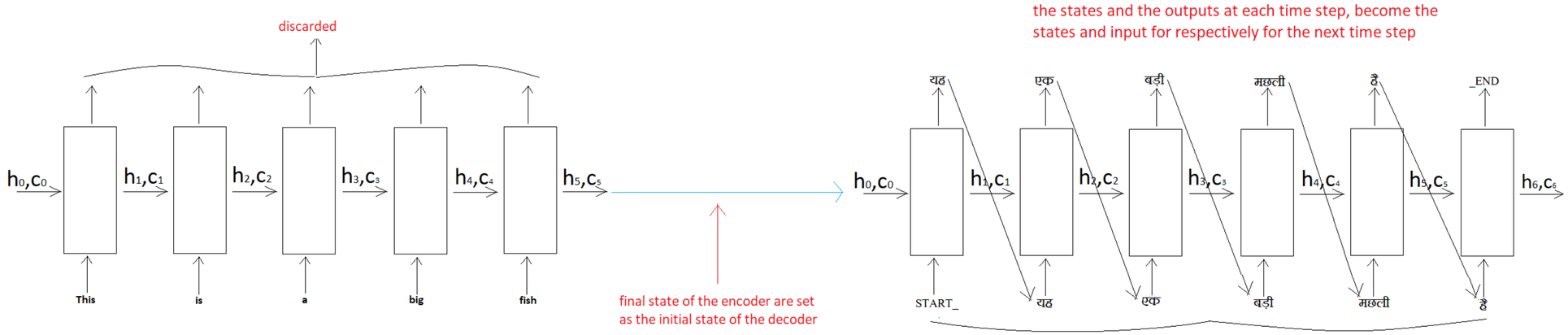
At each time step, we preserve the states of the decoder. The predicted output is fed as input in the next time step

The loop breaks when the decoder predicts the _END

SEQ-SEQ MULTILAYER LSTM

Encoder-Decoder Architecture

Decoder LSTM - Inference Mode

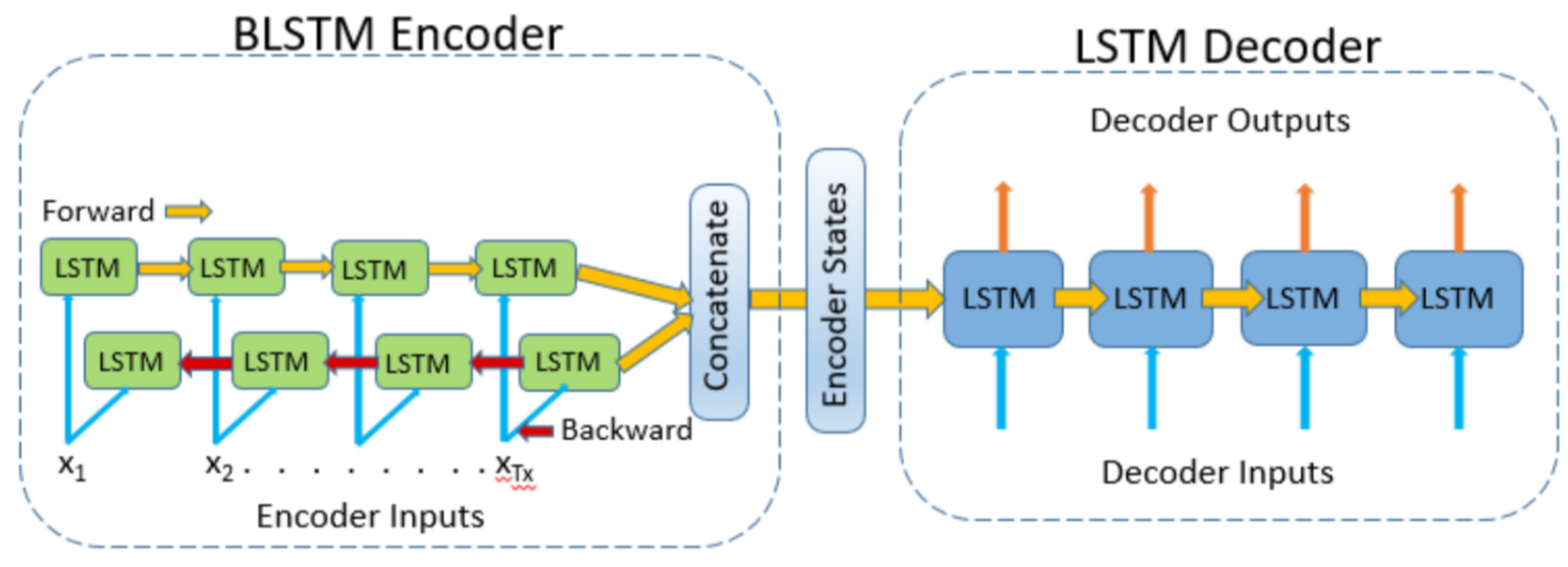


SUMMARY OF THE INFERENCE PROCESS

TEXT TRANSLATION

Seq-Seq Bidirectional LSTM

Encoder-Decoder Architecture



Encoder-Decoder Architecture

2 recurrent components: Forward & Backward

The forward component computes the hidden and cell states like a standard LSTM

The backward component computes the hidden and cell states by taking the input sequence in a reverse-chronological order

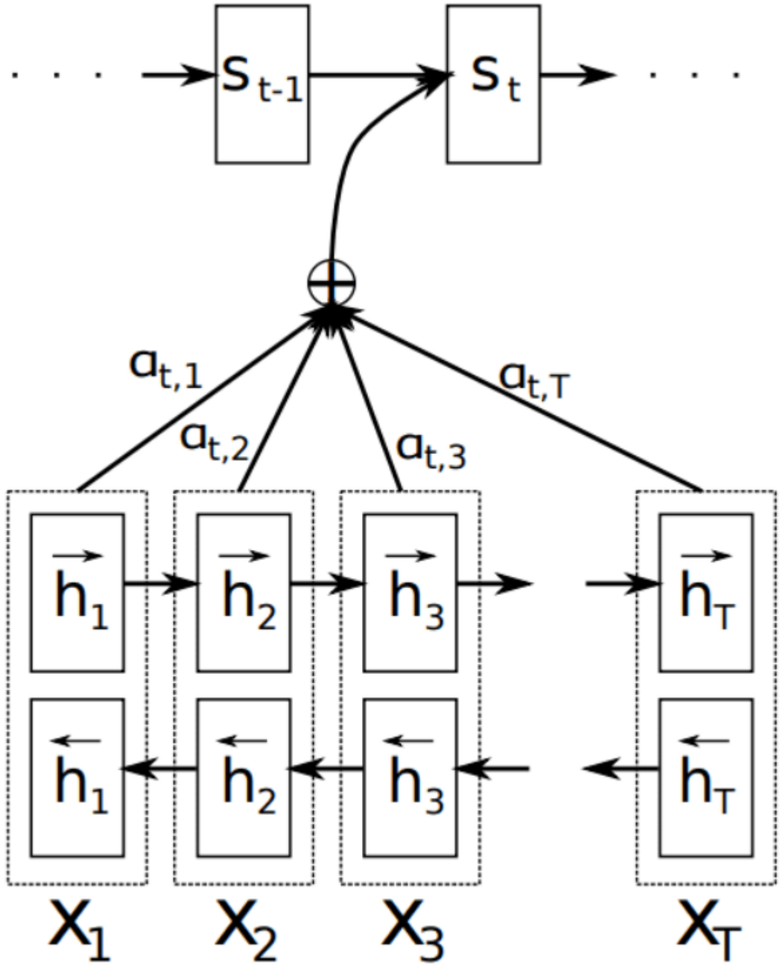
This creates a way for the network to see the future and learn its weight accordingly, allowing to capture some dependencies.

For correct encoding, the forward component have to be concatenated with those of the backward component respectively.

TEXT TRANSLATION

Attention Model

Encoder-Decoder Architecture



ATTENTION MODEL

Encoder-Decoder Architecture

Encoder works similar to the Multilayer LSTM model

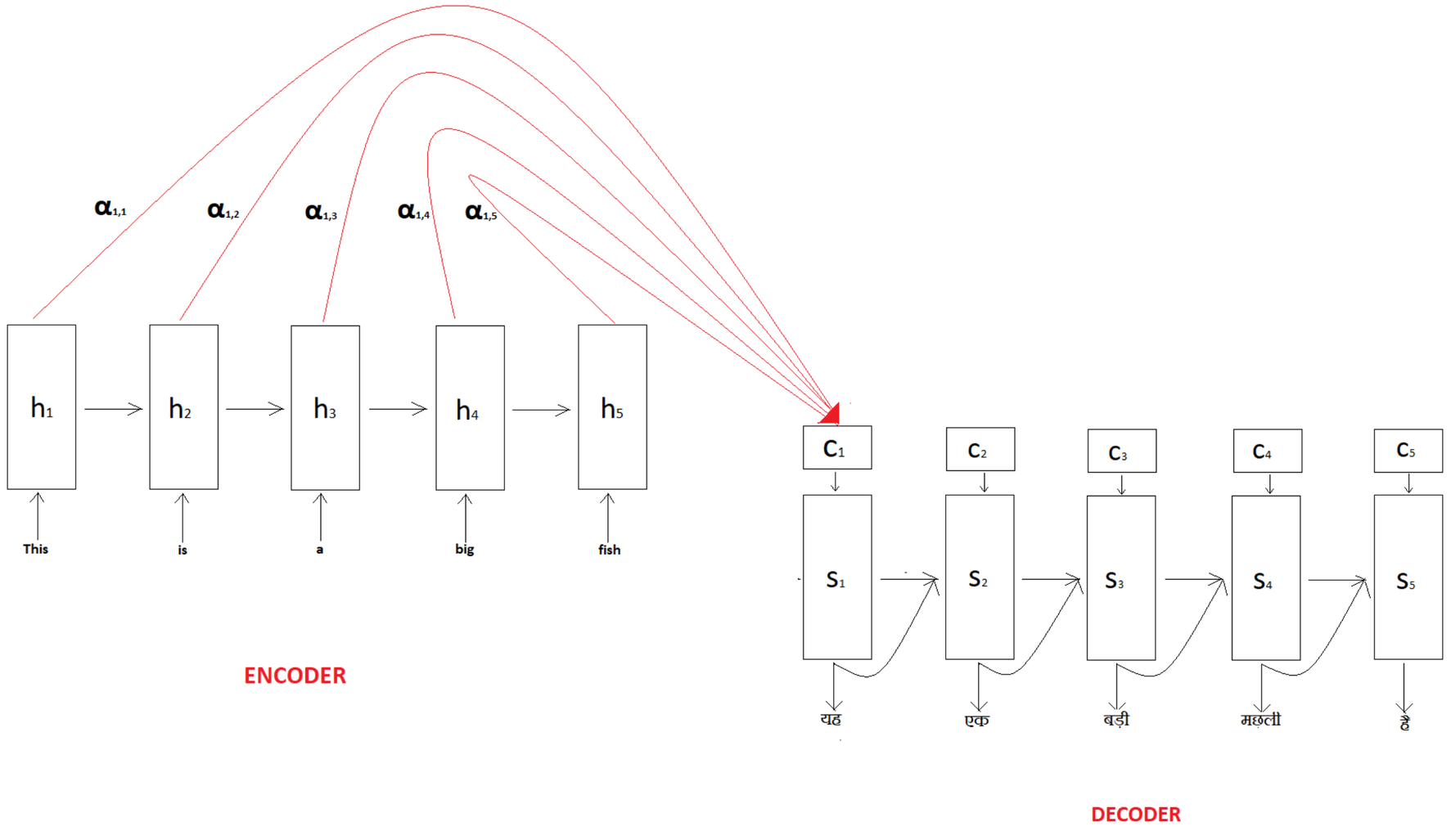
Decoder's hidden state is computed with a context vector, the previous output and the previous hidden state

It also has a seperate context vector c<i> for each target word

$$c^{<i>} = \sum_{t'} \alpha^{<i,t'>} a^{<t'>}$$

ATTENTION MODEL

Encoder-Decoder Architecture



Optical Character Recognition

OCR Pipeline

1. Image Pre-processing
2. Text Detection
3. Text Recognition

OCR

Image Pre-processing

There are 2 main steps

1. Resize the image to new width and height – 320 * 320

2. Construct a blob from the image

OCR

OCR

Text Detection

EAST: Efficient and Accurate Scene Text detection

It is a neural network model, which is trained to directly predict the existence of text instances and their geometries from full images.

This eliminates intermediate steps such as candidate proposal, text region formation and word partition.

OCR

EAST

Pipeline

Image is fed into the FCN and multiple channels of pixel-level text score map and geometry are generated.

The score stands for the confidence of the geometry shape predicted at the same location

There are two geometry shapes for text regions, rotated box (RBOX) and quadrangle (QUAD).

Thresholding is applied to each predicted region. The geometries whose scores are over the threshold is selected

Results after NMS are considered the final output of the pipeline.

EAST

Model

The model can be decomposed in to three parts:

Feature Extractor stem

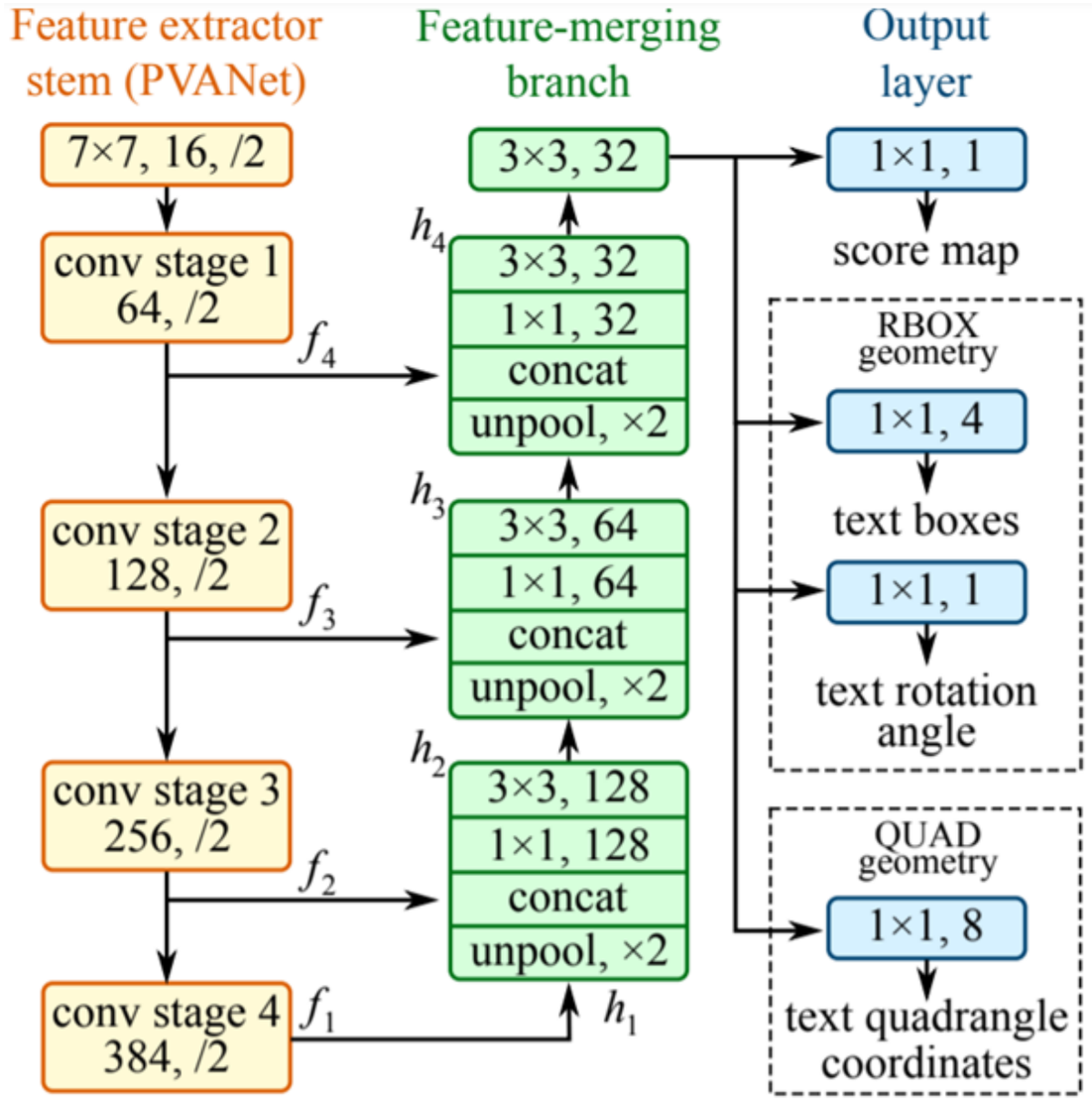
Feature-merging branch

Output layer.

OCR

EAST

Model



OCR

Text Recognition

Tesseract

Tesseract is finding templates in pixels, letters, words and sentences.

It uses two-step approach known as adaptive recognition

It requires one data stage for character recognition

Then the second stage to fulfil any letters, it wasn't insured in, by letters that can match the word or sentence context.

EasyOCR

OCR

Tesseract

Specific Configuration

Tesseract 4

l: English
language chosen

oem: 1 [Neural nets LSTM engine only.]
OCR Engine modes

psm: 8 [Treat the image as a single word.]
Page segmentation modes

CODE SNIPPETS
PRE-PROCESSING

4. CHANGING ALL TO LOWERCASE CHARACTERS

```
# Lowercase all characters
lines['english_sentence']=lines['english_sentence'].apply(lambda x: x.lower())
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: x.lower())
```

5. REMOVING QUOTES

```
# Remove quotes
lines['english_sentence']=lines['english_sentence'].apply(lambda x: re.sub("'", '', x))
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: re.sub("'", '', x))

# Remove quotes
lines['english_sentence']=lines['english_sentence'].apply(lambda x: re.sub("‘", '', x))
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: re.sub("‘", '', x))

# Remove quotes
lines['english_sentence']=lines['english_sentence'].apply(lambda x: re.sub("’", '', x))
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: re.sub("’", '', x))
```

6. REMOVING SPECIAL CHARACTERS

```
exclude = set(string.punctuation) # Set of all special characters
# Remove all the special characters
lines['english_sentence']=lines['english_sentence'].apply(lambda x: ''.join(ch for ch in x if ch not in exclude))
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: ''.join(ch for ch in x if ch not in exclude))
```

CODE SNIPPETS
PRE-PROCESSING

7. REMOVING DIGITS

8. REMOVING EXTRA SPACES

```
remove_digits = str.maketrans('', '', digits)
lines['english_sentence']=lines['english_sentence'].apply(lambda x: x.translate(remove_digits))
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: x.translate(remove_digits))

lines['hindi_sentence'] = lines['hindi_sentence'].apply(lambda x: re.sub("[२३०८१५७९४६]", "", x))

# Remove extra spaces
lines['english_sentence']=lines['english_sentence'].apply(lambda x: x.strip())
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: x.strip())
lines['english_sentence']=lines['english_sentence'].apply(lambda x: re.sub(" +", " ", x))
lines['hindi_sentence']=lines['hindi_sentence'].apply(lambda x: re.sub(" +", " ", x))
```

9. ADDING < START > AND < END > TOKENS

```
# Add start and end tokens to target sequences
lines['hindi_sentence'] = lines['hindi_sentence'].apply(lambda x : 'START_ '+ x + ' _END')
lines.head()
```

	source	english_sentence	hindi_sentence
0	ted	politicians do not have permission to do what ...	START_ राजनीतिज्ञों के पास जो कार्य करना चाहिए...
1	ted	id like to tell you about one such child	START_ मई आपको ऐसे ही एक बच्चे के बारे में बता...

CODE SNIPPETS
CREATING VOCABULARY

10. CREATING ENGLISH AND HINDI VOCABULARY

```
### Get English and Hindi Vocabulary
all_eng_words=set()
for eng in lines['english_sentence']:
    for word in eng.split():
        if word not in all_eng_words:
            all_eng_words.add(word)

all_hindi_words=set()
for hin in lines['hindi_sentence']:
    for word in hin.split():
        if word not in all_hindi_words:
            all_hindi_words.add(word)

len(all_eng_words)
```

```
16293
```

```
len(all_hindi_words)
```

```
21587
```

CODE SNIPPETS
TOKENIZATION

```
# Differentiating input and target
input_words = sorted(list(all_eng_words))
target_words = sorted(list(all_hindi_words))
num_encoder_tokens = len(all_eng_words)
num_decoder_tokens = len(all_hindi_words)
num_encoder_tokens, num_decoder_tokens
```

```
(16293, 21587)
```

```
num_decoder_tokens += 1 #for zero padding
```

12. TOKENIZATION

```
input_token_index = dict([(word, i+1) for i, word in enumerate(input_words)])
target_token_index = dict([(word, i+1) for i, word in enumerate(target_words)])
```

```
reverse_input_char_index = dict((i, word) for word, i in input_token_index.items())
reverse_target_char_index = dict((i, word) for word, i in target_token_index.items())
```

CODE SNIPPETS
GENERATING BATCHES

3. GENERATING BATCHES

```
def generate_batch(X = X_train, y = y_train, batch_size = 128):
    ''' Generate a batch of data '''
    while True:
        for j in range(0, len(X), batch_size):
            encoder_input_data = np.zeros((batch_size, max_length_src),dtype='float32')
            decoder_input_data = np.zeros((batch_size, max_length_tar),dtype='float32')
            decoder_target_data = np.zeros((batch_size, max_length_tar, num_decoder_tokens),dtype='float32')
            for i, (input_text, target_text) in enumerate(zip(X[j:j+batch_size], y[j:j+batch_size])):
                for t, word in enumerate(input_text.split()):
                    encoder_input_data[i, t] = input_token_index[word] # encoder input seq
                for t, word in enumerate(target_text.split()):
                    if t<len(target_text.split())-1:
                        decoder_input_data[i, t] = target_token_index[word] # decoder input seq
                    if t>0:
                        # decoder target sequence (one hot encoded)
                        # does not include the START_ token
                        # Offset by one timestep
                        decoder_target_data[i, t - 1, target_token_index[word]] = 1.
            yield([encoder_input_data, decoder_input_data], decoder_target_data)
```

CODE SNIPPETS
S2S MULTILAYER LSTM
ENCODER-DECODER TRAIN

4. ENCODER-DECODER ARCHITECTURE

```
latent_dim=500
```

```
# Encoder
encoder_inputs = Input(shape=(None,))
enc_emb_layer =  Embedding(num_encoder_tokens + 1, latent_dim, mask_zero = True)
enc_emb = enc_emb_layer(encoder_inputs)
encoder_lstm = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(enc_emb)

# We discard `encoder_outputs` and only keep the states.
encoder_states = [state_h, state_c]
```

```
# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(num_decoder_tokens + 1, latent_dim, mask_zero = True)
dec_emb = dec_emb_layer(decoder_inputs)

# We set up our decoder to return full output sequences,
# and to return internal states as well. We don't use the
# return states in the training model, but we will use them in inference.
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(dec_emb,
                                     initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model that will turn
# `encoder_input_data` & `decoder_input_data` into `decoder_target_data`
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

CODE SNIPPETS
S2S MULTILAYER LSTM
ENCODER-DECODER TEST

1. ENCODER-DECODER FOR OUTPUT

```
# Encode the input sequence to get the "thought vectors"
encoder_model = Model(encoder_inputs, encoder_states)

# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

dec_emb2= dec_emb_layer(decoder_inputs) # Get the embeddings of the decoder sequence

# To predict the next word in the sequence, set the initial states to the states from the previous time step
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=decoder_states_inputs)
decoder_states2 = [state_h2, state_c2]
decoder_outputs2 = decoder_dense(decoder_outputs2) # A dense softmax layer to generate prob dist. over the target vocabulary

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs2] + decoder_states2)
```

CODE SNIPPETS
S2S MULTILAYER LSTM
ENCODER-DECODER TEST

```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    states_value = encoder_model.predict(input_seq)
    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))
    # Populate the first character of target sequence with the start character.
    target_seq[0, 0] = target_token_index['START_']

    # Sampling loop for a batch of sequences
    # (to simplify, here we assume a batch of size 1).
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += ' '+sampled_char

        # Exit condition: either hit max length
        # or find stop character.
        if (sampled_char == '_END' or
           len(decoded_sentence) > 50):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update states
        states_value = [h, c]

    return decoded_sentence
```

CODE SNIPPETS
S2S MULTILAYER LSTM
PREDICTIONS

```
k=random.randint(0,X_train.shape[0])
(input_seq, actual_output), _ = next(train_gen)
decoded_sentence = decode_sequence(input_seq)
candidate = decoded_sentence
refer = y_train[k:k+1].values[0][6:-4]
bleu5 = nltk.translate.bleu_score.sentence_bleu(references=[refer, refer], hypothesis=candidate, weights=(0.25,0.25,0.25,0.25))

print('Input English sentence:', X_train[k:k+1].values[0])
print('Actual Hindi Translation:', y_train[k:k+1].values[0][6:-4])
print('Predicted Hindi Translation:', decoded_sentence[:-4])
print(bleu5)
```

```
Input English sentence: now let me take you back to parol
Actual Hindi Translation:  अब मैं आपको वापस परोल में लेके चलता हूँ
Predicted Hindi Translation:  मुझे यह बड़ी बुद्दिमानी की बात थी
0.36973125629436004
```

CODE SNIPPETS
S2S BIDIRECTIONAL LSTM
ENCODER-DECODER TRAIN

4. ENCODER-DECODER ARCHITECTURE

```
latent_dim=500
```

```
# Encoder
encoder_inputs = Input(shape=(None,))

#enc_emb =  Embedding(num_encoder_tokens, vec_len, mask_zero = True)(encoder_inputs)
#enc_emb =  Embedding(num_encoder_tokens, vec_len)(encoder_inputs)
enc_emb = Embedding(input_dim = num_encoder_tokens, output_dim = latent_dim, mask_zero = True)(encoder_inputs)

encoder_dropout = (TimeDistributed(Dropout(rate = 0.2)))(enc_emb)
encoder_lstm1 = Bidirectional(LSTM(latent_dim, return_sequences=True), merge_mode='sum')(encoder_dropout)
encoder_lstm2 = Bidirectional(LSTM(latent_dim, return_sequences=True), merge_mode='sum')(encoder_lstm1)
encoder_lstm3 = Bidirectional(LSTM(latent_dim, return_sequences=True), merge_mode='sum')(encoder_lstm2)
encoder_LSTM4_layer = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder_LSTM4_layer(encoder_lstm3)
# We discard `encoder_outputs` and only keep the states.
encoder_states = [state_h, state_c]
```

```
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(num_decoder_tokens, latent_dim, mask_zero = True)
dec_emb = dec_emb_layer(decoder_inputs)
decoder_dropout = (TimeDistributed(Dropout(rate = 0.2)))(dec_emb)
# We set up our decoder to return full output sequences,
# and to return internal states as well. We don't use the
# return states in the training model, but we will use them in inference.


decoder_LSTM_layer = LSTM(latent_dim, return_sequences=True)
decoder_LSTM = decoder_LSTM_layer(decoder_dropout, initial_state = encoder_states)

decoder_LSTM_2_layer = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs,_,_ = decoder_LSTM_2_layer(decoder_LSTM)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model that will turn
# `encoder_input_data` & `decoder_input_data` into `decoder_target_data`
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

CODE SNIPPETS
S2S BIDIRECTIONAL LSTM
ENCODER-DECODER TEST

1. ENCODER-DECODER FOR OUTPUT

```
# Encode the input sequence to get the "thought vectors"
encoder_model = Model(encoder_inputs, encoder_states)

# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

dec_emb2= dec_emb_layer(decoder_inputs) # Get the embeddings of the decoder sequence
decoder_dropout2 = (TimeDistributed(Dropout(rate = 0.2)))(dec_emb2)
decoder_LSTM2 = decoder_LSTM_layer(decoder_dropout2, initial_state = decoder_states_inputs)
decoder_outputs2, state_h2, state_c2 = decoder_LSTM_2_layer(decoder_LSTM2)
decoder_states2 = [state_h2, state_c2]
decoder_outputs2 = decoder_dense(decoder_outputs2) #A dense softmax layer to generate prob dist. over the target vocabulary


# Final decoder model
decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs2] + decoder_states2)
```

CODE SNIPPETS
S2S BIDIRECTIONAL LSTM ENCODER-DECODER TEST

```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    states_value = encoder_model.predict(input_seq)
    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))
    # Populate the first character of target sequence with the start character.
    target_seq[0, 0] = target_token_index['START_']

    # Sampling loop for a batch of sequences
    # (to simplify, here we assume a batch of size 1).
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += ' '+sampled_char

        # Exit condition: either hit max length
        # or find stop character.
        if (sampled_char == '_END' or
           len(decoded_sentence) > 98):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update states
        states_value = [h, c]

    return decoded_sentence
```

CODE SNIPPETS
S2S BIDIRECTIONAL LSTM
PREDICTIONS

```
k=random.randint(0,X_train.shape[0])
(input_seq, actual_output), _ = next(train_gen)
decoded_sentence = decode_sequence(input_seq)
candidate = decoded_sentence
refer = y_train[k:k+1].values[0][6:-4]
bleu6 = nltk.translate.bleu_score.sentence_bleu(references=[refer, refer], hypothesis=candidate, weights=(0.25,0.25,0.25,0.25))

print('Input English sentence:', X_train[k:k+1].values[0])
print('Actual Hindi Translation:', y_train[k:k+1].values[0][6:-4])
print('Predicted Hindi Translation:', decoded_sentence[:-4])
print(bleu6)
```

```
Input English sentence: to protest something
Actual Hindi Translation:  किसी बात के विरोध में
Predicted Hindi Translation:  और सकता भरा हुआ उठाने की
0.4687405329188213
```

CODE SNIPPETS
ATTENTION
ENCODER-DECODER

```python
class encoder(nn.Module):

  def __init__(self, input_size, embedding_size, hidden_size, layers, bidirectional):
    '''
    input_size = size of vocab
    embedding_size = embedding dim
    hidden_size = hidden state size
    layer = num of layers of lstms
    '''
    super().__init__()
    self.embed = nn.Embedding(num_embeddings=input_size, embedding_dim=embedding_size) # output size = (*,embedding_size)
    self.lstm = nn.LSTM(input_size=embedding_size, hidden_size= hidden_size, num_layers=layers, batch_first = True, bidirectional = bidirectional)
    self.bidirectional = bidirectional
    self.fc_hidden = nn.Linear(hidden_size*2, hidden_size)
    self.fc_cell = nn.Linear(hidden_size*2, hidden_size)

  def forward(self,x):
    '''
    x shape = [batch_size, sentence]
    one complete sentence represents a "sequence"
    '''
    x = self.embed(x) # shape [batch_size,  sentence, embed_size]
    output, (hidden_state, cell_state) = self.lstm(x) #shape [batch_size, seq_len, num_directions(2)*hidden_size]

    if self.bidirectional:  #since we have 2 directions so add(concat) hidden of both directions into one
      hidden = torch.cat((hidden_state[0:1], hidden_state[1:2]), dim=2)
      cell = torch.cat((cell_state[0:1], cell_state[1:2]), dim = 2) #output [1(layer), batch, hidden_size*2]
      hidden_state = self.fc_hidden(hidden)
      cell_state = self.fc_cell(cell)

    return output, hidden_state, cell_state
```

CODE SNIPPETS
ATTENTION
ENCODER-DECODER

```python
class decoder(nn.Module):

  def __init__(self,input_size, embedding_size, hidden_size, layers):
    '''
    same configuration as encoder
    here input_size = size of hindi vocab
    '''
    super().__init__()
    self.embed = nn.Embedding(num_embeddings=input_size, embedding_dim=embedding_size) # output size = (*,embedding_size)
    self.lstm = nn.LSTM(input_size=embedding_size, hidden_size= hidden_size, num_layers=layers, batch_first = True)
    self.fc = nn.Linear(in_features=hidden_size, out_features=input_size) #since output would be prob distribution among hindi vocab therefore out_feature=input_size

  def forward(self,x,hidden_state, cell_state):
    '''
    seq_len would be 1 as input is  one word not the whole sentence
    x = [batch_size] ->required-> [batch_size, 1] (1 is seq_len)
    '''
    # print(x.shape)
    x = x.reshape(-1,1) # shape [batch, 1]
    # print(x.shape)
    x = self.embed(x) # shape [batch, 1, embed_dim]

    output, (hidden_state, cell_state) = self.lstm(x, (hidden_state, cell_state)) # shape output=>[batch, 1, hidden_size], hidden=>[layers, batch, hidden_size]
    output = self.fc(output) # shape [batch, 1, hindi_vocab_size]

    #removing extra dim
    output = output.squeeze(dim=1) #shape [batch, hindi_vocab_size]

    return output, hidden_state, cell_state
```

CODE SNIPPETS
ATTENTION
ENCODER-DECODER

```python
class AttnDecoder(nn.Module):
  def __init__(self, input_size, embedding_size, hidden_size, layers):
    super().__init__()

    self.embed = nn.Embedding(num_embeddings=input_size, embedding_dim=embedding_size) # output size = (*,embedding_size)
    self.lstm = nn.LSTM(input_size=hidden_size*2 + embedding_size, hidden_size= hidden_size, num_layers=layers, batch_first = True)
    self.fc = nn.Linear(in_features=hidden_size, out_features=input_size) #since output would be prob distribution among hindi vocab therefore out_feature=input_size

    #encoder_states from encoder => [batch, seq_len(35), 2*hidden_size]
    #prev decoder hidden_state => [batch, layers(1)*directions(2), hidden_size] =>need to be in => [batch, seq(35), hidden_size]
    #therefore input of energy will be along hidden_size ie input = hidden_size*2
    self.energy =  nn.Linear(hidden_size*3, 1) #out [batch, seq_len, 1] (2 hidden state from bidirectional encoder and 1 from prev decoder hidden state => 1+2= 3 hidden states as input)
    self.softmax = nn.Softmax(dim=1)# doing softmax for each word ie (dim=1)


  def forward(self, x, hidden_state, cell_state, encoder_states):
    # print(encoder_states.shape)
    seq_len = encoder_states.shape[1]
    batch_size = encoder_states.shape[0]
    hidden_size = encoder_states.shape[2]

    h_new = hidden_state.repeat(seq_len, 1, 1) #shape [seq_len*1, batch, hidden_size*2(bidirectional)] it will repeat dim=0 seq length times
    #by doing .repeat operation we can concat hidden state with all timestamps of encoder_states
    # print(h_new.shape, encoder_states.shape, hidden_state.shape)
    h_new = h_new.permute(1,0,2) #[batch, seq_len, hidden_size*2]
    energy = self.energy(torch.cat((h_new, encoder_states), dim=2))#input [batch, seq_len(35), hidden_size*3]  out = [batch, seq_len(35), 1]
    att_weights = self.softmax(energy)
    att_weights = att_weights.permute(0,2,1) # [batch, 1, seq_len]

    context = torch.bmm(att_weights, encoder_states) #[batch, 1, hidden_size*2]


    x = x.reshape(-1,1) # shape [batch, 1]
    x = self.embed(x) # shape [batch, 1, embed_dim]

    input_new = torch.cat((context, x), dim=2) #[batch, 1, hidden_size*2 +embed_dim]

    output, (hidden_state, cell_state) = self.lstm(input_new, (hidden_state, cell_state)) # shape output=>[batch, 1, hidden_size], hidden=>[layers, batch, hidden_size]
    output = self.fc(output) # shape [batch, 1, hindi_vocab_size]

    output = output.squeeze(dim=1) #shape [batch, hindi_vocab_size]
    del h_new
    del context
    del input_new
    return output, hidden_state, cell_state
```

CODE SNIPPETS
ATTENTION
ENCODER-DECODER

```python
class seq2seq(nn.Module):
  def __init__(self, encoder, decoder):
    super().__init__()
    self.encoder = encoder
    self.decoder = decoder


  def forward(self, input, target, teaching_force=0.5):
    '''
    input = batch of english sentences[batch, sentece(padded)]
    target = batch of hindi sentences [batch, sentence(padded)]
    '''
    batch_size = input.shape[0]
    seq_len = target.shape[1]
    hindi_vocab_size = Hindi_vocab.vocab_size

    output = torch.zeros((seq_len, batch_size, hindi_vocab_size)).to(device)

    _, hidden, cell = self.encoder(input)
    target = target.permute(1,0) # shape [seq, batch]
    x = target[0] # <START> token

    for i in range(1, seq_len):
      out, hidden, cell = self.decoder(x, hidden, cell) #out shape = [batch, vocab_size]
      output[i] = out
      decoder_guess = out.argmax(1)#max value(confidence)  shape = [batch of words]

      if random.random() < teaching_force:
        x = target[i]
      else:
        x =  decoder_guess

    return output  #shape[seq_len, batch_size, vocab_size]
```

CODE SNIPPETS
ATTENTION
ENCODER-DECODER

```python
class Attnseq2seq(nn.Module):
  def __init__(self, encoder, att_decoder):
    super().__init__()
    self.encoder = encoder
    self.decoder = att_decoder


  def forward(self, input, target, teaching_force=0.5):
    '''
    input = batch of english sentences[batch, sentece(padded)]
    target = batch of hindi sentences [batch, sentence(padded)]
    '''
    batch_size = input.shape[0]
    seq_len = target.shape[1]
    hindi_vocab_size = Hindi_vocab.vocab_size

    output = torch.zeros((seq_len, batch_size, hindi_vocab_size)).to(device)

    encoder_states, hidden, cell = self.encoder(input)
    target = target.permute(1,0) # shape [seq, batch]
    x = target[0] # <START> token

    for i in range(1, seq_len):
      out, hidden, cell = self.decoder(x, hidden, cell, encoder_states) #out shape = [batch, vocab_size]
      output[i] = out
      decoder_guess = out.argmax(1)# taking the word with max value(confidence)  shape = [batch of words]

      if random.random() < teaching_force:
        x = target[i]
      else:
        x =  decoder_guess

    return output  #shape[seq_len, batch_size, vocab_size]
```

CODE SNIPPETS ATTENTION PREDICTIONS

```python
def prediction(x):
    for idx in x:
      if idx == 0:
        break
      print(English_vocab.idx2word[int(idx)],end=' ')

    print()

    x = x.long().reshape(1,-1).to(device)
    ans = translate(x)
    res = []
    for id in ans:
      res.append(Hindi_vocab.idx2word[id])

    return res
```

```python
def translate(input):
      #input = batch of english sentences[batch, sentece(padded)]
      with torch.no_grad():
        guess = []
        encoder_states, hidden, cell = model.encoder(input)
        # x = torch.ones((1)).float().to(device) # <START> token
        x = torch.ones((1)).long().to(device)
        while True:
          out, hidden, cell = model.decoder(x, hidden, cell, encoder_states) #out shape = [batch, vocab_size]
          x = out.argmax(1)# taking the word with max value(confidence)  shape = [batch of words]
          guess.append(int(x[0].detach().cpu()))

          if x == 2:
            break

      return guess
```

CODE SNIPPETS ATTENTION PREDICTIONS

```
k=random.randint(0,10000)
print('Input English sentence:',end=' ')
candidate = prediction(dataset[k][0])
refer = df.iloc[k,1][1:]
bleu2 = nltk.translate.bleu_score.sentence_bleu(references=[refer, refer], hypothesis=candidate, weights=(0.25,0.25,0.25,0.25))

print('Actual Hindi Translation:', refer)
print('Predicted Hindi Translation:', candidate)
print(bleu2)
```

```
Input English sentence: and you have people who guide those computers <END>
Actual Hindi Translation: ['लोग', 'हैं', 'उन', 'कंप्यूटरों', 'के', 'मार्गदर्शन', '<END>']
Predicted Hindi Translation: ['लोग', 'हैं', 'उन', 'कंप्यूटरों', 'के', 'मार्गदर्शन', '<END>']
1.0
```

CODE SNIPPETS
OCR
EAST

```python
## Returns a bounding box and probability score if it is more than minimum confidence
def predictions(prob_score, geo):
  (numR, numC) = prob_score.shape[2:4]
  boxes = []
  confidence_val = []

  # loop over rows
  for y in range(0, numR):
    scoresData = prob_score[0, 0, y]
    x0 = geo[0, 0, y]
    x1 = geo[0, 1, y]
    x2 = geo[0, 2, y]
    x3 = geo[0, 3, y]
    anglesData = geo[0, 4, y]

    # loop over the number of columns
    for i in range(0, numC):
      if scoresData[i] < args["min_confidence"]:
        continue

      (offX, offY) = (i * 4.0, y * 4.0)

      # extracting the rotation angle for the prediction and computing the sine and cosine
      angle = anglesData[i]
      cos = np.cos(angle)
      sin = np.sin(angle)

      # using the geo volume to get the dimensions of the bounding box
      h = x0[i] + x2[i]
      w = x1[i] + x3[i]

      # compute start and end for the text pred bbox
      endX = int(offX + (cos * x1[i]) + (sin * x2[i]))
      endY = int(offY - (sin * x1[i]) + (cos * x2[i]))
      startX = int(endX - w)
      startY = int(endY - h)

      boxes.append((startX, startY, endX, endY))
      confidence_val.append(scoresData[i])

  # return bounding boxes and associated confidence_val
  return (boxes, confidence_val)
```

CODE SNIPPETS
OCR
EAST

```
# loop over the bounding boxes to find the coordinate of bounding boxes
for (startX, startY, endX, endY) in boxes:
  # scale the coordinates based on the respective ratios in order to reflect bounding box on the original image
  startX = int(startX * rW)
  startY = int(startY * rH)
  endX = int(endX * rW)
  endY = int(endY * rH)

  #extract the region of interest
  r = orig[startY:endY, startX:endX]

  #configuration setting to convert image to string.
  configuration = ("--oem 1 --psm 8")

  ##This will recognize the text from the image of bounding box
  text = pytesseract.image_to_string(r, lang='eng', config=configuration)

  # append bbox coordinate and associated text to the list of results
  results.append(((startX, startY, endX, endY), text))

#Display the image with bounding box and recognized text
orig_image = orig.copy()
```

CODE SNIPPETS
OCR
TESSERACT

USING TESSERACT

```
!pip install langdetect
```

```
def imageToTextTamil(path):
  image_path_in_colab=path
  extractedInformation = pytesseract.image_to_string(Image.open(image_path_in_colab),lang='tam')
  print(extractedInformation)

  from langdetect import detect_langs
  print(detect_langs(extractedInformation))
```

CODE SNIPPETS
OCR
EASYOCR

USING EASYOCR

```
!pip install easyocr --no-deps
!pip install python-bidi

import cv2
from google.colab.patches import cv2_imshow
```

```
import easyocr
reader = easyocr.Reader(['ta','en'])
```

```
def printImage(path):
  img = cv2.imread(path, cv2.IMREAD_UNCHANGED)
  cv2_imshow(img)
```

```
def imageToTextTamil_1(path):
  printImage(path)
  bounds = reader.readtext(path)
  for i in range(len(bounds)):
    print(bounds[i][1])
```

CODE SNIPPETS
FINAL PREDICTION

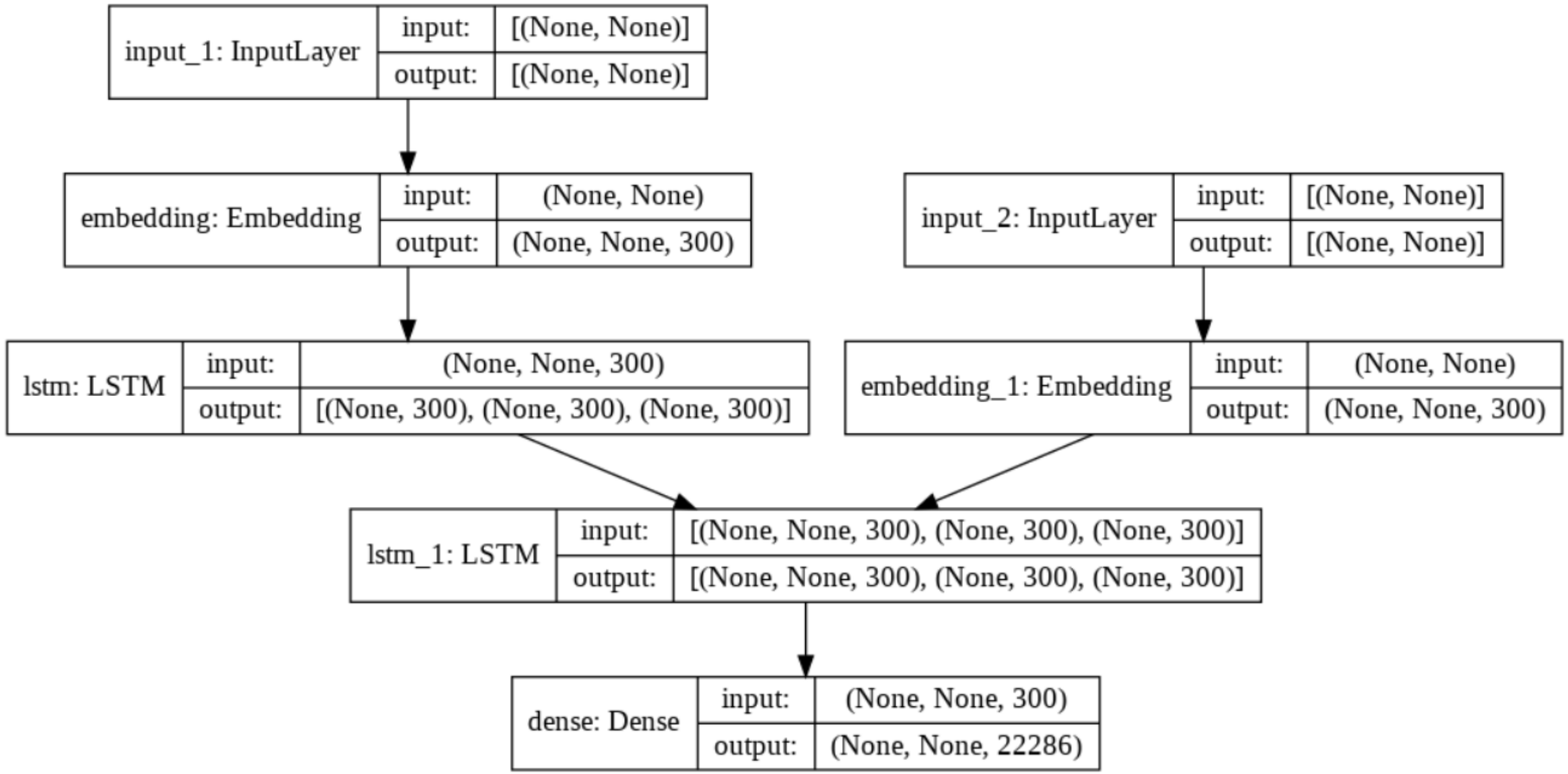
```
def get_prediction(val):
    keys=np.array([])
    val=val.split()
    for word in val:
        word=word.lower()
        for key, value in English_vocab.idx2word.items():
            if word == value:
                keys=np.append(keys,key)
    keys=np.append(keys,1)
    while(len(keys)!=37):
        keys=np.append(keys,0)
    target=torch.tensor(keys,dtype=torch.float32)
    print(prediction(target))
```

Seq-Seq Multilayer LSTM

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
embedding (Embedding)           (None, None, 300)    5203800     input_1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 300)    6686100     input_2[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 300), (None, 721200      embedding[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, None, 300),  721200      embedding_1[0][0]
                                                                 lstm[0][1]
                                                                 lstm[0][2]
__________________________________________________________________________________________________
dense (Dense)                   (None, None, 22286)  6708086     lstm_1[0][0]
==================================================================================================
Total params: 20,040,386
Trainable params: 20,040,386
Non-trainable params: 0
```

MODEL SUMMARY

Seq-Seq Multilayer LSTM



MODEL PLOT

Seq-Seq Bidirectional LSTM

MODEL SUMMARY

```
Model: "model"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_1 (InputLayer)         [(None, None)]            0
_________________________________________________________________________________________________
embedding (Embedding)        (None, None, 500)         8147000     input_1[0][0]
_________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 500)      0           embedding[0][0]
_________________________________________________________________________________________________
bidirectional (Bidirectional)  (None, None, 500)       4004000     time_distributed[0][0]
_________________________________________________________________________________________________
input_2 (InputLayer)         [(None, None)]            0
_________________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, None, 500)      4004000     bidirectional[0][0]
_________________________________________________________________________________________________
embedding_1 (Embedding)      (None, None, 500)         10794000    input_2[0][0]
_________________________________________________________________________________________________
bidirectional_2 (Bidirectional) (None, None, 500)      4004000     bidirectional_1[0][0]
_________________________________________________________________________________________________
time_distributed_1 (TimeDistrib (None, None, 500)      0           embedding_1[0][0]
_________________________________________________________________________________________________
lstm_3 (LSTM)                [(None, 500), (None,      2002000     bidirectional_2[0][0]
_________________________________________________________________________________________________
lstm_4 (LSTM)                (None, None, 500)         2002000     time_distributed_1[0][0]
                                                                   lstm_3[0][1]
                                                                   lstm_3[0][2]
_________________________________________________________________________________________________
lstm_5 (LSTM)                [(None, None, 500),       2002000     lstm_4[0][0]
_________________________________________________________________________________________________
dense (Dense)                (None, None, 21588)       10815588    lstm_5[0][0]
=================================================================================================
Total params: 47,774,588
Trainable params: 47,774,588
Non-trainable params: 0
```

Seq-Seq Bidirectional LSTM

MODEL PLOT

Attention Model

```
Attnseq2seq(
  (encoder): encoder(
    (embed): Embedding(16978, 256)
    (lstm): LSTM(256, 256, batch_first=True, bidirectional=True)
    (fc_hidden): Linear(in_features=512, out_features=256, bias=True)
    (fc_cell): Linear(in_features=512, out_features=256, bias=True)
  )
  (decoder): AttnDecoder(
    (embed): Embedding(20376, 256)
    (lstm): LSTM(768, 256, batch_first=True)
    (fc): Linear(in_features=256, out_features=20376, bias=True)
    (energy): Linear(in_features=768, out_features=1, bias=True)
    (softmax): Softmax(dim=1)
  )
)
```

MODEL SUMMARY

Results

TEXT TRANSLATION

Architecture	No. of HiddenLSTM	BLEU Score
Multilayer LSTM	300	0.3389
	500	0.4915
Bidirectional LSTM	300	0.4350
	500	0.4995
Attention	256	1.0

Seq-seq Multilayer LSTM

TRANSLATION PREDICTIONS

Input English sentence: when i do my work
Actual Hindi Translation: जब मैं अपना काम करता हूँ
Predicted Hindi Translation: हमने वैज्ञानिक क्रांतियाँ उन्होंने शुरुवात किया।

Input English sentence: now let me take you back to parol
Actual Hindi Translation: अब मैं आपको वापस परोल में लेके चलता हूँ
Predicted Hindi Translation: मुझे यह बड़ी बुद्दिमानी की बात थी

Input English sentence: the incremental advances have added up to something
Actual Hindi Translation: वृद्धिशील विकास का नतीजा इस तरह का है
Predicted Hindi Translation: हम उस समय ब्रांक्स में रहा करते थे

Seq-seq Bidirectional LSTM

TRANSLATION PREDICTIONS

Input English sentence: next thing you knew theres a full page covering us positively
Actual Hindi Translation: और पलक झपकते ही हमें पूरे पेज की बढिया कवरेज मिली थी
Predicted Hindi Translation: ये बहत से ज्यादा से अधिक से ज्यादा से बड़ी ख़ुद एक प्रतिदिन

Input English sentence: to protest something
Actual Hindi Translation: किसी बात के विरोध में
Predicted Hindi Translation: और सकता भरा हुआ उठाने की

Input English sentence: this was something really really revolutionary
Actual Hindi Translation: यह सच में बहुत क्रांतिकारी था
Predicted Hindi Translation: लेकिन वाली चलता संबंधी

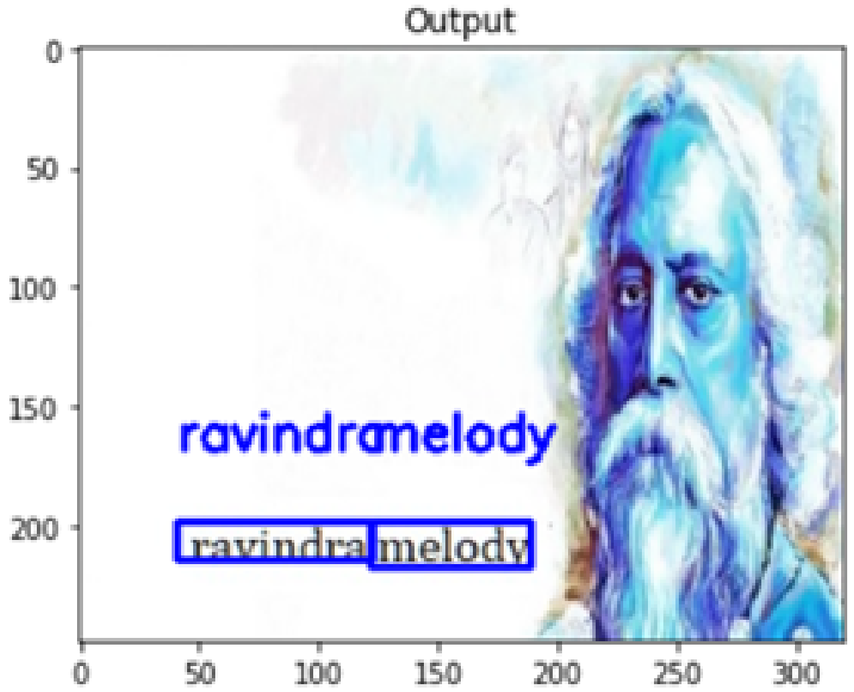
Attention Model

TRANSLATION PREDICTIONS

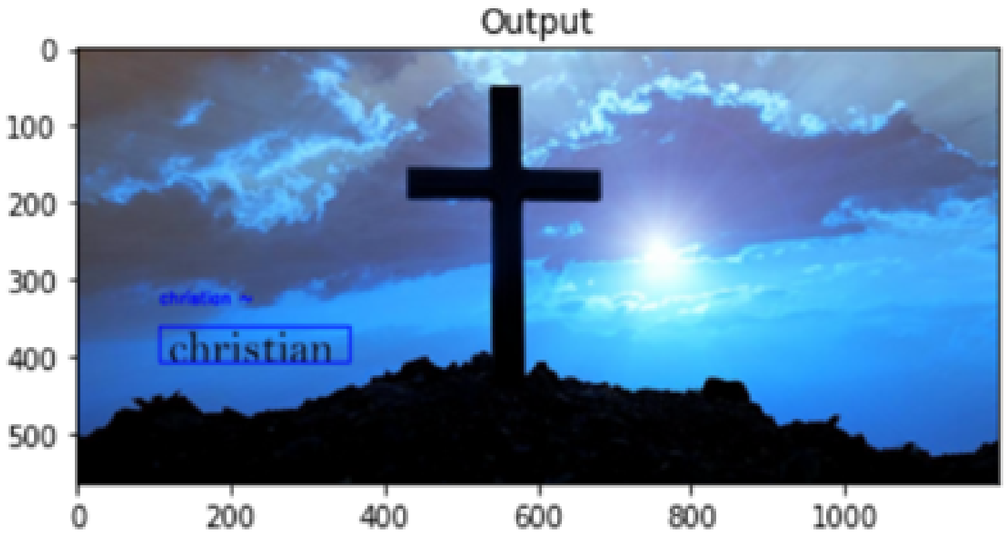
```
Input English sentence: a student missed 18 <END>
Actual Hindi Translation: ['एक', 'बच्चे', 'ने', '१८', 'गलत', 'जवाब', 'दिये', '<END>']
Predicted Hindi Translation: ['एक', 'बच्चे', 'ने', '१८', 'गलत', 'जवाब', 'दिये', '<END
>']

Input English sentence: and you have people who guide those computers <END>
Actual Hindi Translation: ['लोग', 'हैं', 'उन', 'कंप्यूटरों', 'के', 'मार्गदर्शन', '<END>']
Predicted Hindi Translation: ['लोग', 'हैं', 'उन', 'कंप्यूटरों', 'के', 'मार्गदर्शन', '<END>']

Input English sentence: passable government <END>
Actual Hindi Translation: ['कामचलाऊ', 'सरकार', '<END>']
Predicted Hindi Translation: ['कामचलाऊ', 'सरकार', '<END>']
```

EAST

```
ravindra

melody
```




```
christian ~
```



OCR PREDICTIONS

TESSERACT Tamil

த கை: ல் 1
எல்லா பெண்களையும் விருலடித்து

ம ப ய ரம?
ட்ப 2 சட்டம பனதடப

[ta:0.9999999885219631]

இதுகான் சுவறான ஸனிசயம்-.-

ட ஆ. 10810%0₹08.0எ

[ta:0.9999999999975504]

OCR PREDICTIONS

EasyOCR Tamil

எல்லா பெண்களையும் விசிலடித்து
திரும்பி பார்க்க வைத்தாலும் செருப்படி
வாங்காத ஒரே ஜீவன் குக்கர் தான்!!!

இதுதான் தவறான விசயம்
rorlphotoscom

OCR PREDICTIONS

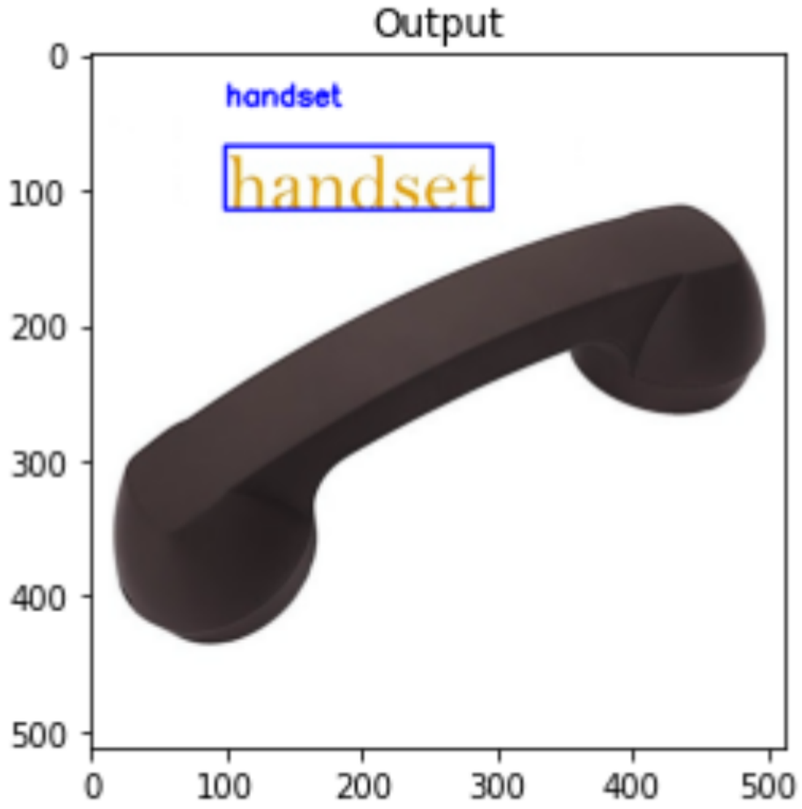
Nanonets Tamil [FOR COMPARISON ONLY]

எல்லா பெண்களையும் விசிலடித்து
திரும்பி பார்க்க வைத்தாலும் செருப்படி
வாங்காத ஒரே ஜீவன் குக்கர் தான்

இதுதான்
தவறான
விசெயம்..

OCR PREDICTIONS

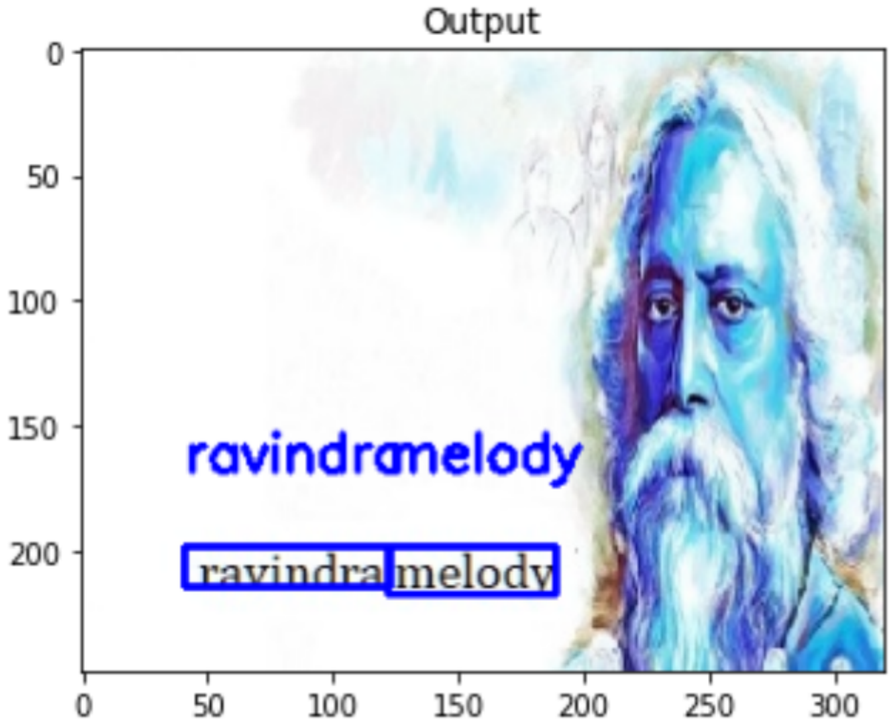
handset



handset <END>
['हैंडसेट', '<END>']

ravindra

melody



ravindra melody <END>
['रवीन्द्र', 'संगीत', '<END>']

FINAL MODEL PREDICTIONS

Conclusion

Implemented a neural networks to work with three Machine Translation Architecture. They utilize different Encoder-Decoder Model to predict the translation from one language to another.

Have worked with different kinds of method to build of the Encoder-Decoder Model.

Have Visualized the different stages of Encoding and Decoding the sentences.

Implemented OCR, using 3 techniques: EAST, Tessract and EasyOCR. Learnt in depth about Computer Vision


Thank You