MINI PROJECT REPORT

A report submitted in partial fulfilment of the requirements for the Award of Degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS and COMMUNICATION ENGINEERING

by

Nitish Kumar Jha

(02151202819)

Vivek Dubey

(01951202819)

Shilpa Gautam

(02551202819)

under the supervision of

Mr. Abhishek

DEPARTMENT OF ECE BHARATI VIDYAPEETH'S COLLEGE OF ENGINEERING
Affiliated to Guru Gobind Singh Indraprastha University, New Delhi

CANDIDATE’s DECLARATION

We, the undersigned, solemnly declare that the Mini project report is based on our own work carried out during the course of our study under the supervision of **Mr. Abhishek**.

We assert the statements made and the conclusions drawn are an outcome of my research work. We further certify that the work contained in the report is original and has been done by us under the general supervision of my supervisor. Moreover, the work has not been submitted to any other institution for any other degree/diploma/certificate in this university or any other University of India or abroad. Also, whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and giving their details in the references

Nitish Kumar Jha (02151202819)

Vivek Dubey (01951202819)

Shilpa Gautam (02551202818)

ACKNOWLEDGEMENT

We express my deep gratitude to Mr. Abhishek, for his valuable guidance and suggestions throughout our training. We would like to extend my sincere thanks to Head of the Department, Dr. Kirti Gupta for her time-to-time suggestions to complete our project work. We are also thankful to Dr. Dharmendra Saini, Principal for providing me the facilities to carry out my project work.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

If you talk to a man in a language he understands, that goes to his head. If you talk to him in his own language, that goes to his heart." – Nelson Mandela.

The beauty of language transcends boundaries and cultures. Learning a language other than our mother tongue is a huge advantage. But the path to bilingualism, or multilingualism, can often be a long, never-ending one.

There are so many little nuances that we get lost in the sea of words. Things have, however, become so much easier with online translation services (I'm looking at you Google Translate!). The objective is to convert a English sentence to its french counterpart using a Neural Machine Translation (NMT) system. The language translation model that we are going to develop will translate English sentences into their French language counterparts. To develop such a model, we need a dataset that contains English sentences and their French translations.

.

CHAPTER ONE INTRODUCTION

Most of us were introduced to machine translation when Google came up with the service. But the concept has been around since the middle of last century. Research work in Machine Translation (MT) started as early as 1950's, primarily in the United States. These early systems relied on huge bilingual dictionaries, hand-coded rules, and universal principles underlying natural language.

In 1954, IBM held a first ever public demonstration of a machine translation. The system had a pretty small vocabulary of only 250 words and it could translate only 49 hand-picked Russian sentences to English. The number seems minuscule now but the system is widely regarded as an important milestone in the progress of machine translation.

Soon, two schools of thought emerged:

- Empirical trial-and-error approaches, using statistical methods, and
- Theoretical approaches involving fundamental linguistic research

In 1964, the Automatic Language Processing Advisory Committee (ALPAC) was established by the United States government to evaluate the progress in Machine Translation. ALPAC did a little prodding around and published a report in November 1966 on the state of MT. Below are the key highlights from that report:

- It raised serious questions on the feasibility of machine translation and termed it hopeless
- Funding was discouraged for MT research
- It was quite a depressing report for the researchers working in this field
- Most of them left the field and started new careers.

Not exactly a glowing recommendation!

1.1 INCORPORATED PACKAGES

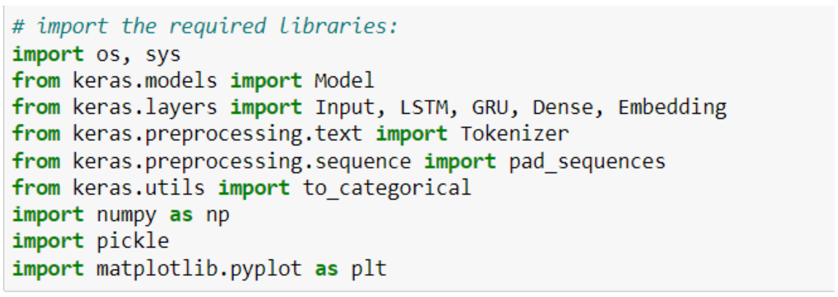
1.1.1 NUMPY

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be Performed. NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open-source project.

1.1.2 KERAS

Keras gives fundamental reflections and building units for creation and transportation of ML arrangements with high iteration velocity. It takes full advantage of the scalability and crossplatform capabilities of TensorFlow. The core data structures of Keras are layers and models. All the layers used in the CNN model are implemented using Keras. Along with the conversion of the class vector to the binary class matrix in data processing, it helps to compile the overall model. We can see in the Fig 1 how to import pacages.



```
# import the required libraries:
import os, sys
from keras.models import Model
from keras.layers import Input, LSTM, GRU, Dense, Embedding
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
import numpy as np
import pickle
import matplotlib.pyplot as plt
```

Fig 1: Imported Packages

1.1.3 Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats .
- Embed in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib.

CHAPTER TWO

METHODOLOGY

2.1 DATA PRE-PROCESSING

Data pre-processing involves conversion of data from a given format to much more user friendly, desired and meaningful format. It can be in any form like tables, images, videos, graphs, etc. This organized information fit in with an information model or composition and captures relationship between different entities . We need to generate two copies of the translated sentence: one with the start-of-sentence token and the other with the end-of-sentence token.

2.1.1 TOKENIZATION AND PADDING

The next step is tokenizing the original and translated sentences and applying padding to the sentences that are longer or shorter than a certain length, which in case of inputs will be the length of the longest input sentence. And for the output this will be the length of the longest sentence in the output. For tokenization, the Tokenizer class from the keras.preprocessing.text library can be used. The tokenizer class performs two tasks:

1. It divides a sentence into the corresponding list of word.
2. Then it converts the words to integers.

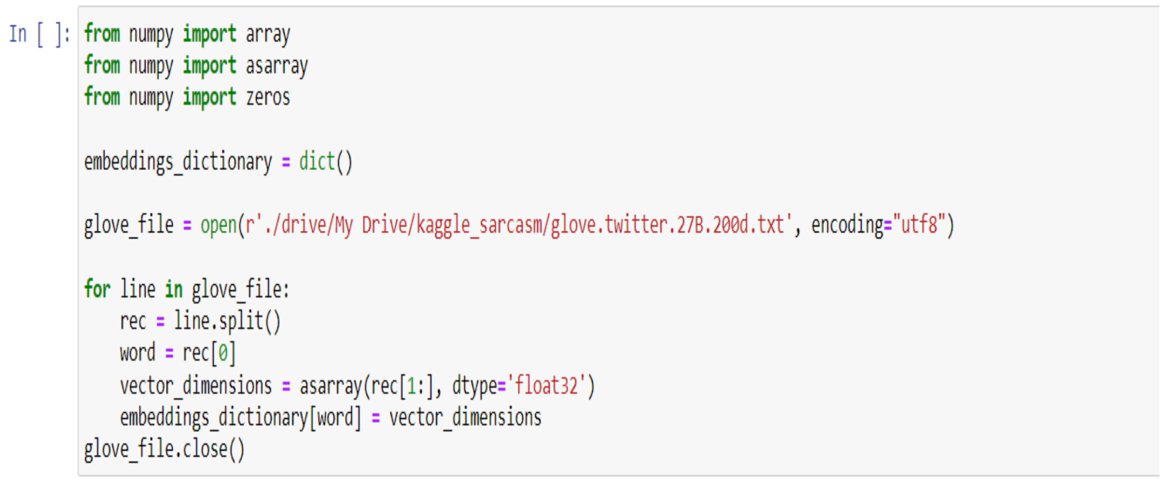
Also the **word_index** attribute of the Tokenizer class returns a word-to-index dictionary where words are the keys and the corresponding integers are the values.

2.1.2 WORD EMBEDDINGS

We already converted our words into integers. So what's the difference between integer representation and word embeddings?There are two main differences between single integer representation and word embeddings. With integer reprensentation, a word is represented only with a single integer. With vector representation a word is represented by a vector of 50, 100, 200, or whatever dimensions you like. Hence, word embeddings capture a lot more information about words. Secondly, the single-integer representation doesn't capture the relationships between different words. On the contrary, word embeddings retain relationships between the words. You can either use custom word embeddings or you can use pretrained word embeddings.

For English sentences, i.e. the inputs, we will use the GloVe word embeddings. For the translated French sentences in the output, we will use custom word embeddings.

Let's create word embeddings for the inputs first. To do so, we need to load the GloVe word vectors into memory. We will then create a dictionary where words are the keys and the corresponding vectors are values,



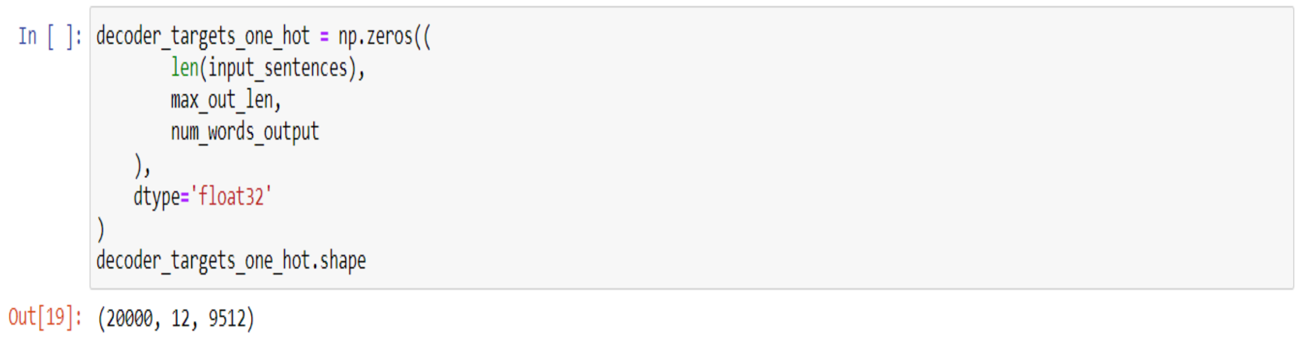
```
In [ ]: from numpy import array
        from numpy import asarray
        from numpy import zeros

        embeddings_dictionary = dict()

        glove_file = open(r'./drive/My Drive/kaggle_sarcasm/glove.twitter.27B.200d.txt', encoding="utf8")

        for line in glove_file:
            rec = line.split()
            word = rec[0]
            vector_dimensions = asarray(rec[1:], dtype='float32')
            embeddings_dictionary[word] = vector_dimensions
        glove_file.close()
```

Fig 2: Procedure for Word embeddings

2.2 TRAINING OF MODEL

2.2.1 CREATING THE MODEL USING CNN ARCHITECTURE

The first thing we need to do is to define our outputs, as we know that the output will be a sequence of words. Recall that the total number of unique words in the output are 9511. Therefore, each word in the output can be any of the 9511 words. The length of an output sentence is 12. And for each input sentence, we need a corresponding output sentence. Therefore, the final shape of the output will be:

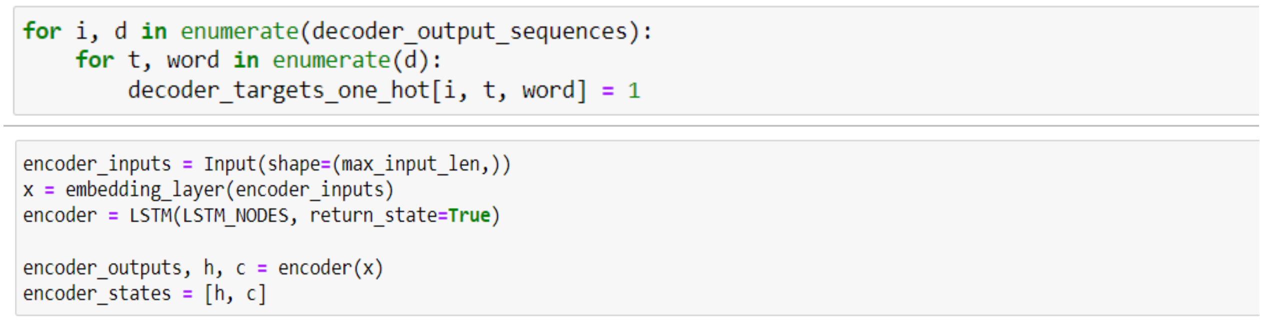


```
In [ ]: decoder_targets_one_hot = np.zeros((
                len(input_sentences),
                max_out_len,
                num_words_output
            ),
            dtype='float32'
        )
        decoder_targets_one_hot.shape

Out[19]: (20000, 12, 9512)
```

Fig 3: Creating Model

2.2.2 TRAINING THE MODEL

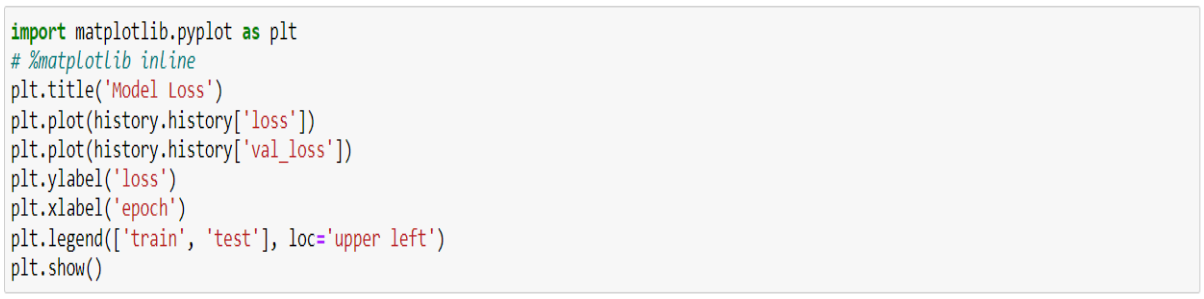
To make predictions, the final layer of the model will be a dense layer, therefore we need the outputs in the form of one-hot encoded vectors, since we will be using softmax activation function at the dense layer. To create such one-hot encoded output, the next step is to assign 1 to the column number that corresponds to the integer representation of the word.Next, we need to create the encoder and decoders. The input to the encoder will be the sentence in English and the output will be the hidden state and cell state of the LSTM.



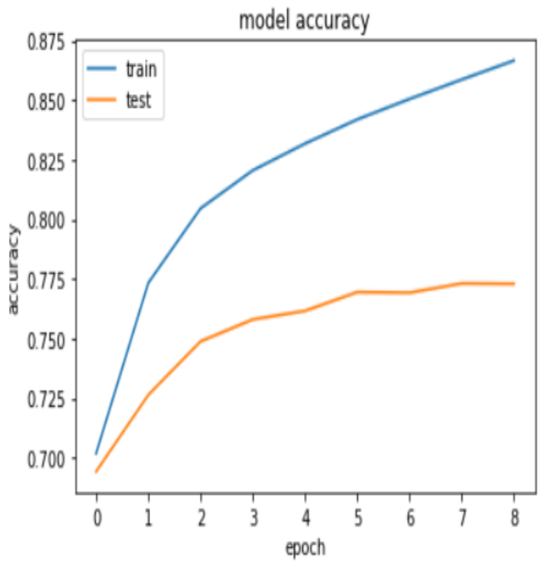
```
for i, d in enumerate(decoder_output_sequences):
    for t, word in enumerate(d):
        decoder_targets_one_hot[i, t, word] = 1
```

```
encoder_inputs = Input(shape=(max_input_len,))
x = embedding_layer(encoder_inputs)
encoder = LSTM(LSTM_NODES, return_state=True)

encoder_outputs, h, c = encoder(x)
encoder_states = [h, c]
```

Figure 4: creating encoder and decoder



```
import matplotlib.pyplot as plt
# %matplotlib inline
plt.title('Model Loss')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



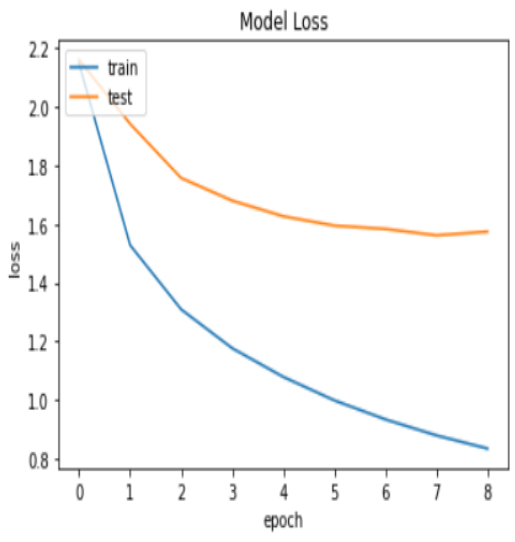


Figure 5: Training and Validation accuracy **Figure 6: Training and Validation loss**

CHAPTER THREE

CONCLUSION

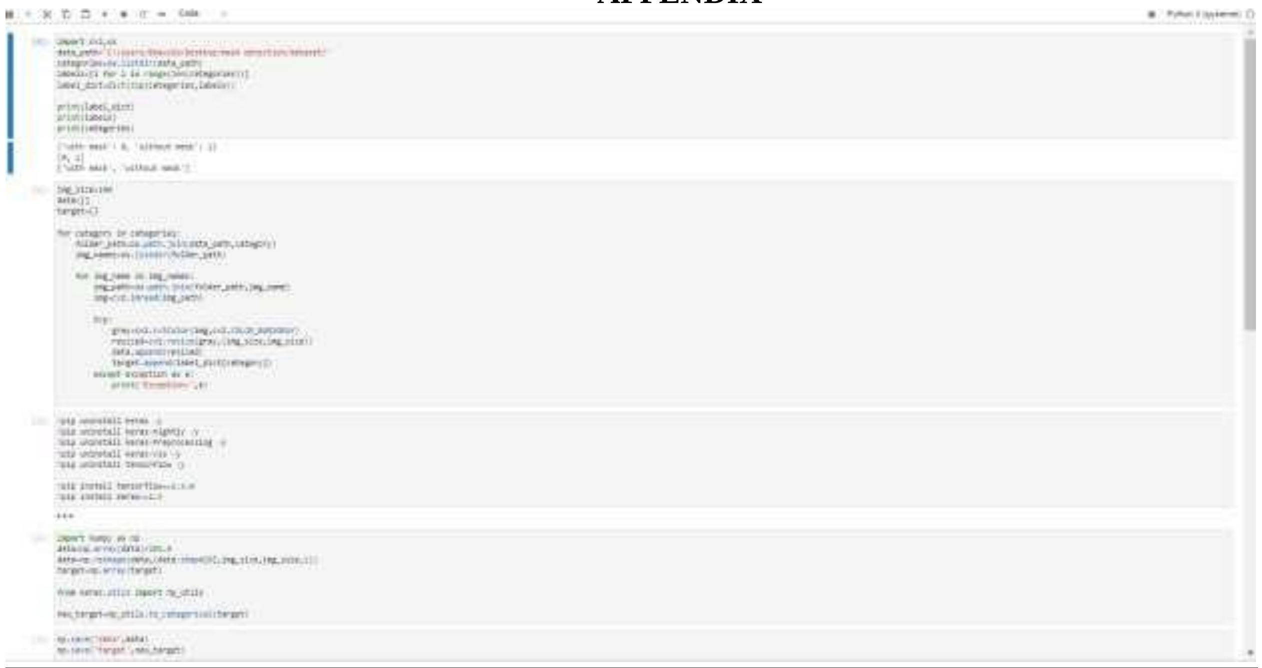
In this report, we briefly explained the motivation of the work at first. Then, we illustrated the learning and performance task of the model. Using basic ML tools and simplified techniques the method has achieved reasonably high accuracy. The method will accept an input-padded sequence English sentence (in the integer form) and will return the translated French sentence. While this Report provides an introduction to NMT using the Encoder Decoder structure, the implemented attention mechanism is rather basic. We Can Extend This to a Real Time Speech to Speech translator i.e, what ever we speak in our language gets translated into targeted language and it would be read aloud as speech.

REFERENCES

- Chollet, F., & others, (2015), Keras, https://keras.io
- Dalal, N., and Triggs, B., Histograms of oriented gradients for human detection, CVPR '05, 2005.
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L., ImageNet: A Large-Scale Hierarchical Image Database, IEEE Computer Vision and Pattern Recognition (CVPR), 2009.
- Deng, J., Guo, J., Ververas, E., Kotsia, I., and Zafeiriou, S., RetinaFace: Single-Shot Multi-Level Face Localisation in the Wild, 2020, IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 2020, pp. 5202-5211, doi: 10.1109/CVPR42600.2020.00525.
- Ejaz, M.S., Islam, M.R., Sifatullah, M., Sarker, A., Implementation of principal component analysis on masked and non-masked face recognition, 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), 2019, pp. 1–5
- Girshick, R., Fast R-CNN, Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1440–1448.

- Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q., Densely Connected Convolutional Networks, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 2261-2269, doi: 10.1109/CVPR.2017.243.
- Redmon, J., Divvala, S., Girshick, R., and Farhadi, A., You Only Look Once: Unified real-time object detection, IEEE Conference ComputerVision and Pattern Recognition (CVPR), 2016.
- Nieto-Rodríguez, A., Mucientes, M., Brea, V.M., (2015), System for Medical Mask Detection in the Operating Room Through Facial Attributes. In: Paredes R., Cardoso J., Pardo X. (eds) Pattern Recognition and Image Analysis. IbPRIA 2015. Lecture Notes in Computer Science, vol 9117. Springer, Cham. https://doi.org/10.1007/978-3-319- 19390-8_16.
- Qin, B., and Li, D., Identifying facemask-wearing condition using image super-resolution with classification network to prevent COVID-19 (2020), Unpublished results, doi: 10.21203/rs.3.rs28668/v1.
- Ren, S., He, K., Girshick, R., & Sun, J., (2015), Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (pp. 91–99), MIT Press.
- M. Turk and A.P. Pentland, "Face Recognition Using Eigenfaces", IEEE Conf. Computer Vision and Pattern Recognition, 1991.

APPENDIX


```
import cv2,os
data_path=[illegible]
categories=os.listdir(data_path)
labels=[i for i in range(len(categories))]
label_dict=dict(zip(categories,labels))

print(label_dict)
print(labels)
print(categories)
```

```
{'with mask': 0, 'without mask': 1}
[0, 1]
['with mask', 'without mask']
```

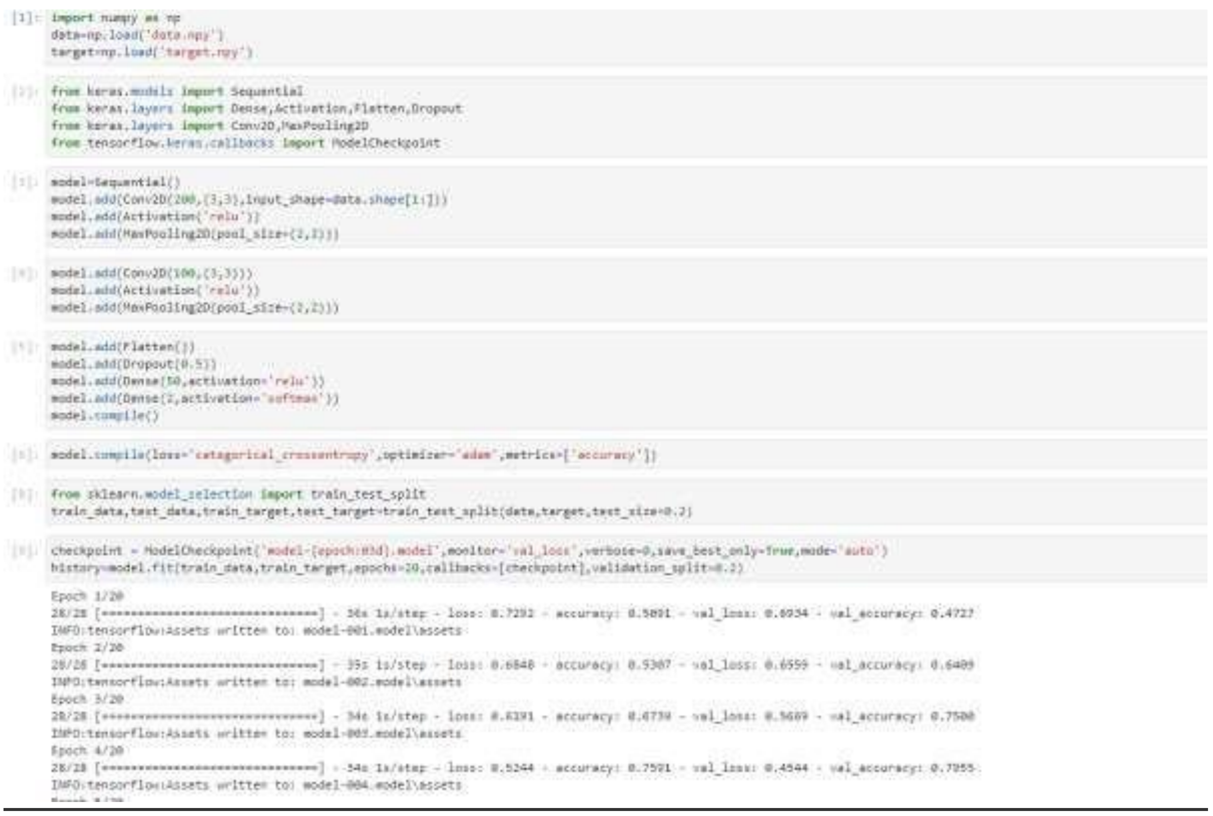
```
img_size=100
data=[]
target=[]

for category in categories:
    folder_path=os.path.join(data_path,category)
    img_names=os.listdir(folder_path)

    for img_name in img_names:
        img_path=os.path.join(folder_path,img_name)
        img=cv2.imread(img_path)

        try:
            gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
            resized=cv2.resize(gray,(img_size,img_size))
            data.append(resized)
            target.append(label_dict[category])
        except Exception as e:
            print('Exception:',e)
```

```
!pip uninstall keras -y
!pip uninstall keras-nightly -y
!pip uninstall keras-Preprocessing -y
!pip uninstall keras-vis -y
!pip uninstall tensorflow -y

!pip install tensorflow==[illegible]
!pip install keras==[illegible]
```

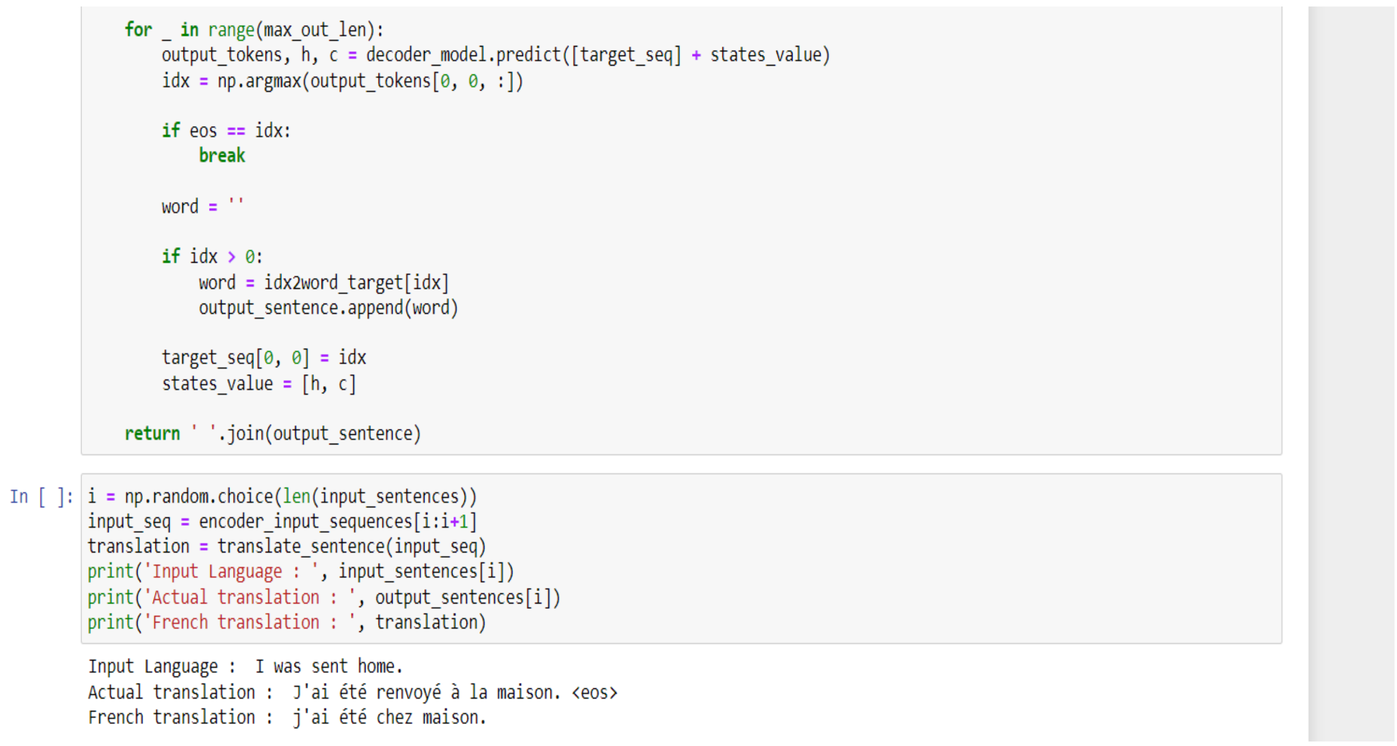
```
...
```

```
import numpy as np
data=np.array(data)/255.0
data=np.reshape(data,(data.shape[0],img_size,img_size,1))
target=np.array(target)

from keras.utils import np_utils

new_target=np_utils.to_categorical(target)
```

```
np.save('data',data)
np.save('target',new_target)
```



```
[1]: import numpy as np
     data=np.load('data.npy')
     target=np.load('target.npy')

[2]: from keras.models import Sequential
     from keras.layers import Dense,Activation,Flatten,Dropout
     from keras.layers import Conv2D,MaxPooling2D
     from tensorflow.keras.callbacks import ModelCheckpoint

[3]: model=Sequential()
     model.add(Conv2D(200,(3,3),input_shape=data.shape[1:]))
     model.add(Activation('relu'))
     model.add(MaxPooling2D(pool_size=(2,2)))

[4]: model.add(Conv2D(100,(3,3)))
     model.add(Activation('relu'))
     model.add(MaxPooling2D(pool_size=(2,2)))

[5]: model.add(Flatten())
     model.add(Dropout(0.5))
     model.add(Dense(50,activation='relu'))
     model.add(Dense(2,activation='softmax'))
     model.compile()

[6]: model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

[7]: from sklearn.model_selection import train_test_split
     train_data,test_data,train_target,test_target=train_test_split(data,target,test_size=0.2)

[8]: checkpoint = ModelCheckpoint('model-{epoch:03d}.model',monitor='val_loss',verbose=0,save_best_only=True,mode='auto')
     history=model.fit(train_data,train_target,epochs=20,callbacks=[checkpoint],validation_split=0.2)

     Epoch 1/20
     28/28 [==============================] - 36s 1s/step - loss: 0.7292 - accuracy: 0.5091 - val_loss: 0.6934 - val_accuracy: 0.4727
     INFO:tensorflow:Assets written to: model-001.model\assets
     Epoch 2/20
     28/28 [==============================] - 35s 1s/step - loss: 0.6848 - accuracy: 0.5307 - val_loss: 0.6559 - val_accuracy: 0.6409
     INFO:tensorflow:Assets written to: model-002.model\assets
     Epoch 3/20
     28/28 [==============================] - 34s 1s/step - loss: 0.6391 - accuracy: 0.6730 - val_loss: 0.5609 - val_accuracy: 0.7500
     INFO:tensorflow:Assets written to: model-003.model\assets
     Epoch 4/20
     28/28 [==============================] - 34s 1s/step - loss: 0.5244 - accuracy: 0.7591 - val_loss: 0.4544 - val_accuracy: 0.7955
     INFO:tensorflow:Assets written to: model-004.model\assets
```



```
    for _ in range(max_out_len):
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)
        idx = np.argmax(output_tokens[0, 0, :])

        if eos == idx:
            break

        word = ''

        if idx > 0:
            word = idx2word_target[idx]
            output_sentence.append(word)

        target_seq[0, 0] = idx
        states_value = [h, c]

    return ' '.join(output_sentence)
```

```
In [ ]: i = np.random.choice(len(input_sentences))
        input_seq = encoder_input_sequences[i:i+1]
        translation = translate_sentence(input_seq)
        print('Input Language : ', input_sentences[i])
        print('Actual translation : ', output_sentences[i])
        print('French translation : ', translation)

        Input Language :  I was sent home.
        Actual translation :  J'ai été renvoyé à la maison. <eos>
        French translation :  j'ai été chez maison.
```

```
28/28 [==============================] - 37s 1s/step - loss: 0.4221 - accuracy: 0.8060 - val_loss: 0.3369 - val_accuracy: 0.8727
INFO:tensorflow:Assets written to: model-005.model\assets
Epoch 6/20
28/28 [==============================] - 41s 1s/step - loss: 0.3074 - accuracy: 0.8852 - val_loss: 0.2447 - val_accuracy: 0.9273
INFO:tensorflow:Assets written to: model-006.model\assets
Epoch 7/20
28/28 [==============================] - 37s 1s/step - loss: 0.2283 - accuracy: 0.9125 - val_loss: 0.2006 - val_accuracy: 0.9364
INFO:tensorflow:Assets written to: model-007.model\assets
Epoch 8/20
28/28 [==============================] - 36s 1s/step - loss: 0.2217 - accuracy: 0.9136 - val_loss: 0.2586 - val_accuracy: 0.8955
Epoch 9/20
28/28 [==============================] - 36s 1s/step - loss: 0.1858 - accuracy: 0.9330 - val_loss: 0.1959 - val_accuracy: 0.9318
INFO:tensorflow:Assets written to: model-009.model\assets
Epoch 10/20
28/28 [==============================] - 39s 1s/step - loss: 0.1503 - accuracy: 0.9466 - val_loss: 0.1330 - val_accuracy: 0.9409
INFO:tensorflow:Assets written to: model-010.model\assets
Epoch 11/20
28/28 [==============================] - 35s 1s/step - loss: 0.1072 - accuracy: 0.9716 - val_loss: 0.1781 - val_accuracy: 0.9409
Epoch 12/20
28/28 [==============================] - 34s 1s/step - loss: 0.0931 - accuracy: 0.9602 - val_loss: 0.1324 - val_accuracy: 0.9455
INFO:tensorflow:Assets written to: model-012.model\assets
Epoch 13/20
28/28 [==============================] - 34s 1s/step - loss: 0.0948 - accuracy: 0.9636 - val_loss: 0.1200 - val_accuracy: 0.9545
INFO:tensorflow:Assets written to: model-013.model\assets
Epoch 14/20
28/28 [==============================] - 34s 1s/step - loss: 0.0883 - accuracy: 0.9659 - val_loss: 0.2015 - val_accuracy: 0.9364
Epoch 15/20
28/28 [==============================] - 34s 1s/step - loss: 0.0579 - accuracy: 0.9830 - val_loss: 0.1248 - val_accuracy: 0.9500
Epoch 16/20
28/28 [==============================] - 34s 1s/step - loss: 0.0528 - accuracy: 0.9841 - val_loss: 0.1709 - val_accuracy: 0.9364
Epoch 17/20
28/28 [==============================] - 34s 1s/step - loss: 0.0559 - accuracy: 0.9830 - val_loss: 0.1436 - val_accuracy: 0.9591
Epoch 18/20
28/28 [==============================] - 34s 1s/step - loss: 0.0453 - accuracy: 0.9852 - val_loss: 0.1074 - val_accuracy: 0.9545
INFO:tensorflow:Assets written to: model-018.model\assets
Epoch 19/20
28/28 [==============================] - 34s 1s/step - loss: 0.0425 - accuracy: 0.9830 - val_loss: 0.1034 - val_accuracy: 0.9727
INFO:tensorflow:Assets written to: model-019.model\assets
Epoch 20/20
28/28 [==============================] - 34s 1s/step - loss: 0.0626 - accuracy: 0.9750 - val_loss: 0.1827 - val_accuracy: 0.9409
```

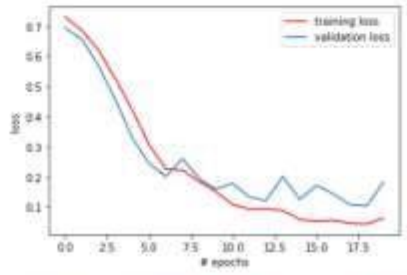
[17]:

```
INFO:tensorflow:Assets written to: model-017.model\assets
```

[18]:

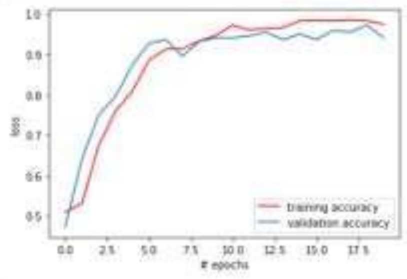
```python
import matplotlib.pyplot as plt
plt.plot(history.history['loss'],'r',label='training loss')
plt.plot(history.history['val_loss'],label='validation loss')
```

```python
plt.plot(history.history['loss'],'r',label='training loss')
plt.plot(history.history['val_loss'],label='validation loss')
plt.xlabel('# epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



[11]:

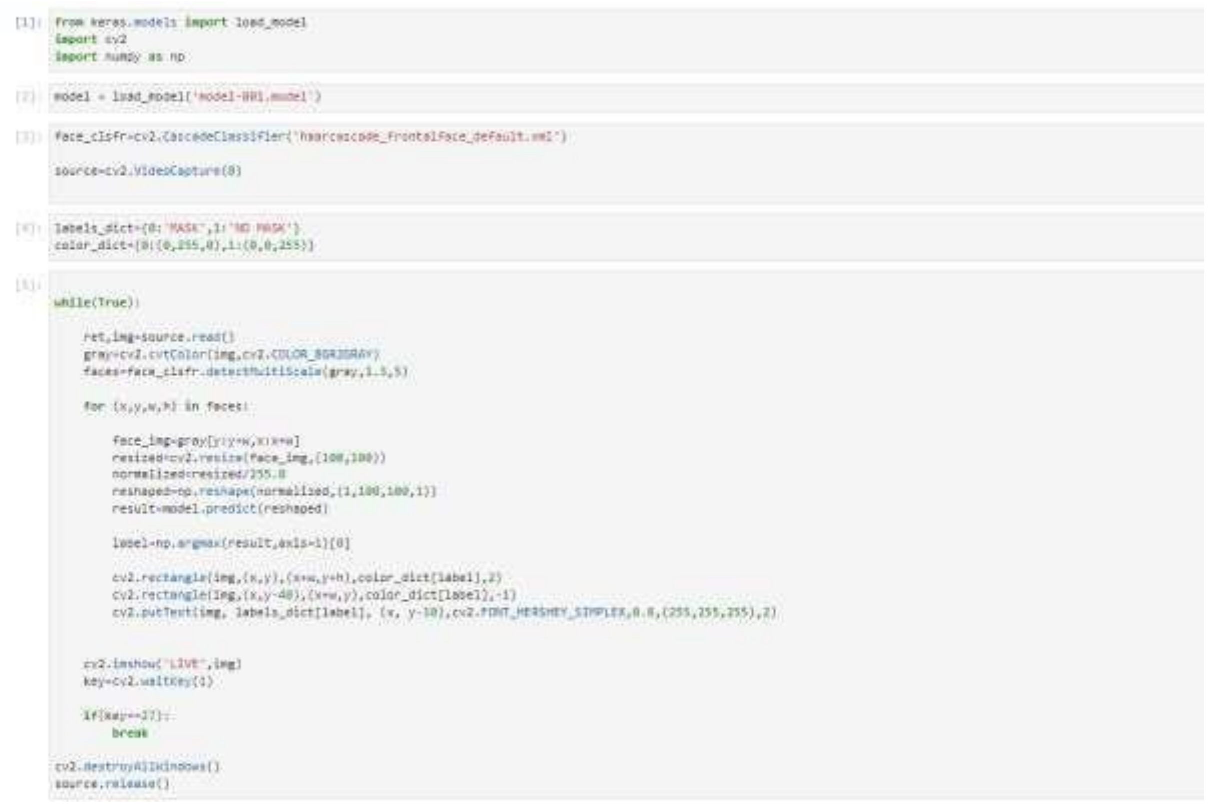
```python
plt.plot(history.history['accuracy'],'r',label='training accuracy')
plt.plot(history.history['val_accuracy'],label='validation accuracy')
plt.xlabel('# epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



[12]:

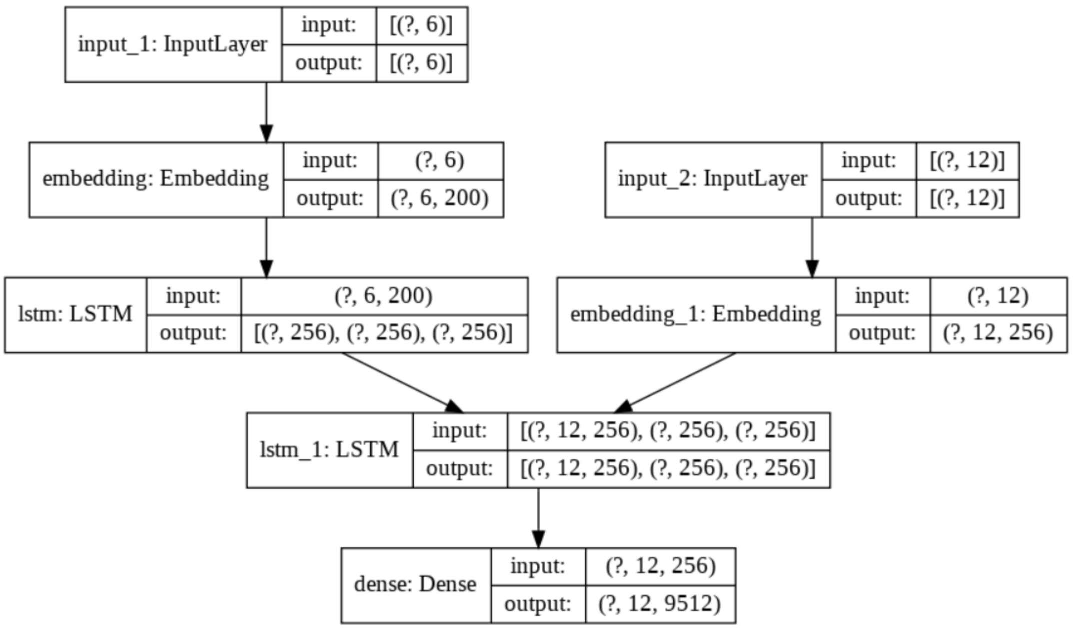
```python
print(model.evaluate(test_data,test_target))
```

```
9/9 [==============================] - 3s 326ms/step - loss: 0.2836 - accuracy: 0.9094
[0.28360557556152344, 0.9094203114509583]
```



```
[1]: from keras.models import load_model
     import cv2
     import numpy as np

[2]: model = load_model('model-001.model')

[3]: face_clsfr=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

     source=cv2.VideoCapture(0)

[4]: labels_dict={0:'MASK',1:'NO MASK'}
     color_dict={0:(0,255,0),1:(0,0,255)}

[5]:
     while(True):

         ret,img=source.read()
         gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
         faces=face_clsfr.detectMultiScale(gray,1.3,5)

         for (x,y,w,h) in faces:

             face_img=gray[y:y+w,x:x+w]
             resized=cv2.resize(face_img,(100,100))
             normalized=resized/255.0
             reshaped=np.reshape(normalized,(1,100,100,1))
             result=model.predict(reshaped)

             label=np.argmax(result,axis=1)[0]

             cv2.rectangle(img,(x,y),(x+w,y+h),color_dict[label],2)
             cv2.rectangle(img,(x,y-40),(x+w,y),color_dict[label],-1)
             cv2.putText(img, labels_dict[label], (x, y-10),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)


         cv2.imshow('LIVE',img)
         key=cv2.waitKey(1)

         if(key==27):
             break

     cv2.destroyAllWindows()
     source.release()
```

Out[25]:


input_1: InputLayer
input: [(?, 6)]
output: [(?, 6)]
embedding: Embedding
input: (?, 6)
output: (?, 6, 200)
input_2: InputLayer
input: [(?, 12)]
output: [(?, 12)]
lstm: LSTM
input: (?, 6, 200)
output: [(?, 256), (?, 256), (?, 256)]
embedding_1: Embedding
input: (?, 12)
output: (?, 12, 256)
lstm_1: LSTM
input: [(?, 12, 256), (?, 256), (?, 256)]
output: [(?, 12, 256), (?, 256), (?, 256)]
dense: Dense
input: (?, 12, 256)
output: (?, 12, 9512)