```
#CNN
# things we need for NLP
import nltk

from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()
from flask import Flask,jsonify, request, render_template,json
app = Flask(__name__)
# things we need for Tensorflow
#%tensorflow_version 1.x
# things we need for Tensorflow
import numpy as np
import tflearn
import tensorflow as tf
import random


# import our chat-bot intents file
#import json
#open the exact location
with open("intents.json") as jd:
    intents = json.load(jd)

#if files is missing download this specific
nltk.download('punkt')

#Create words, classes and documents
words = []
classes = []
documents = []
ignore_words = ['?',',','.','!']

# loop through each sentence in our intents patterns
for intent in intents['intents']:
    for pattern in intent['patterns']:

        # tokenize each word in the sentence

        w = nltk.word_tokenize(pattern)
        # add to our words list
        words.extend(w)
        # add to documents in our corpus
        documents.append((w, intent['tag']))
        # add to our classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])

# stem and lower each word and remove duplicates
words = sorted(list(set([stemmer.stem(w.lower()) for w in words if w
```

```
not in ignore_words])))

# remove duplicates
classes = sorted(list(set(classes)))

# create our training data
training = []
output = []
# create an empty array for our output
output_empty = [0] * len(classes)

# create our training data
training = []
for doc in documents:
    # initialize our bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]
    # stem each word
    pattern_words = [stemmer.stem(word.lower()) for word in
pattern_words]
    # create our bag of words array
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    # output is a '0' for each tag and '1' for current tag
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])

# shuffle our features and turn into np.array
random.shuffle(training)
training = np.array(training)
# create train and test lists
train_x = list(training[:,0])
train_y = list(training[:,1])
# reset underlying graph data
#tf.reset_default_graph()
# Build neural network
net = tflearn.input_data(shape=[None, len(train_x[0])])
net = tflearn.fully_connected(net, len(train_x))
net = tflearn.fully_connected(net, len(train_x))
net = tflearn.fully_connected(net, len(train_y[0]),
activation='softmax')
net = tflearn.regression(net)

# Define model and setup tensorboard
model = tflearn.DNN(net, tensorboard_dir='tflearn_logs')
```

```
# Start training (apply gradient descent algorithm)
model.fit(train_x, train_y, n_epoch=1000, batch_size=8,
show_metric=True)
model.save('model.tflearn')

# save all of our data structures
import pickle
pickle.dump({'words':words,
'classes':classes,'train_x':train_x,'train_y':train_y},open( "training
_data", "wb" ))



data = pickle.load( open( "training_data", "rb" ) )
words = data['words']
classes = data['classes']
train_x = data['train_x']
train_y = data['train_y']
# import our chat-bot intents file
import json
with open('intents.json') as jd:
    intents = json.load(jd)

# load our saved model
model.load('./model.tflearn')

def clean_up_sentence(sentence):
    # tokenize the pattern
    sentence_words = nltk.word_tokenize(sentence)
    # stem each word
    sentence_words = [stemmer.stem(word.lower()) for word in
sentence_words]
    return sentence_words

# return bag of words array: 0 or 1 for each word in the bag that
exists in the sentence
def bow(sentence, words):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bag of words
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                bag[i] = 1

    return(np.array(bag))

ERROR_THRESHOLD = 0.5
```

```
def classify(sentence):
    # generate probabilities from the model
    results = model.predict([bow(sentence, words)])[0]
    # filter out predictions below a threshold
    results = [[i,r] for i,r in enumerate(results) if
r>ERROR_THRESHOLD]
    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []
    for r in results:
        return_list.append((classes[r[0]], r[1]))
    # return tuple of intent and probability
    return return_list

def response(sentence):
    results = classify(sentence)
    # if we have a classification then find the matching intent tag
    if results:
        # loop as long as there are matches to process
        while results:
            for i in intents['intents']:
                # find a tag matching the first result
                if i['tag'] == results[0][0]:
                    # a random response from the intent
                    return (random.choice(i['responses']))

            results.pop(0)




@app.route("/")
def hello():
    return render_template("index.html")
@app.route('/op', methods=['POST'])
def sum_num():

    rf=request.form

    for key in rf.keys():
        data=key


    data_dic=json.loads(data)
    input_sent=response(data_dic['str'])
    print(data_dic.keys())



    resp_dic={'str':str(input_sent)}
```

```
    resp = jsonify(resp_dic)
    resp.headers['Access-Control-Allow-Origin']='*'
    return resp

if __name__=='__main__':
    app.run()
```