

Prototype of a Conversational Chatbot with built with a Natural Language Toolkit

A Chatbot built with Convolutional Neural Networks for a fictitious pizza restaurant called ReadyPizza



By :

Faith Haruna

Fontys University of Applied Sciences, Eindhoven, The Netherlands

Contents

1	Introduction	3
2	Methods	3
	a) Setting up Git	3
	b) Setting up and activating a deployment virtual environment.....	4
	c) Setting up project directory structure	5
	i. Dataset	5
	ii. Images	6
	iii. Model Artifacts	6
	d) Scripting	7
	i. Requirements Text file	7
	ii. Pre-processor Python file	7
	iii. App Python file.....	8
	e) Deploying with Streamlit.....	9
3	Results.....	10
4	Conclusion.....	11
5	References.....	11

1 Introduction

The motivation of this project is to demo a Conversational chatbot built with NLTK and Tensorflow on a collection of linguistic data compiled as written texts.

This project utilizes cutting-edge tools, frameworks and libraries such as Tensorflow, Streamlit tools to build an intelligent conversational chatbot centered around a pizza ordering restaurant called ReadyPizza.

2 Methods

a) Setting up Git

Git was used for version control to manage all the files and folders for easier future updates and changes. A repository called “PizzaRestaurantChatbot”¹ was created in GitHub from which a cloned local copy will be used for development.

```
MINGW64~/c/Users/Harun/NLP/chatbot
Harun@LAPTOP-IDJ190NR MINGW64 ~/NLP/chatbot (master)
$ echo "# PizzaRestaurantChatbot" >> README.md && git init
Initialized empty Git repository in C:/Users/Harun/NLP/chatbot/.git/

Harun@LAPTOP-IDJ190NR MINGW64 ~/NLP/chatbot (master)
$ git add README.md
warning: LF will be replaced by CRLF in README.md.
The file will have its original line endings in your working directory

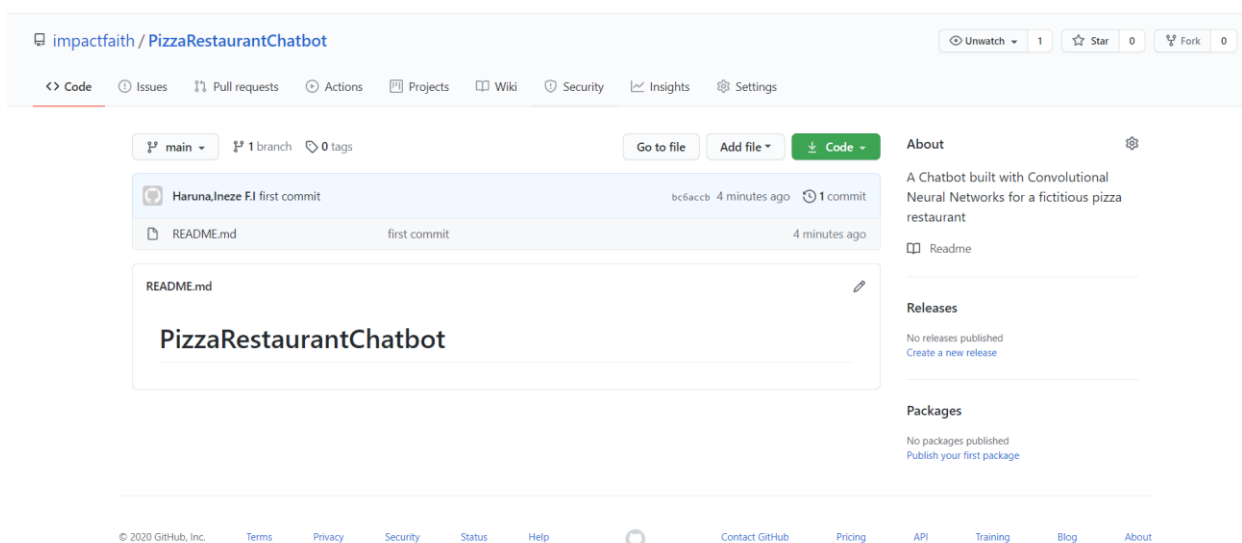
Harun@LAPTOP-IDJ190NR MINGW64 ~/NLP/chatbot (master)
$ git commit -m "first commit"
[master (root-commit) bc6acbb] first commit
1 file changed, 1 insertion(+)
create mode 100644 README.md

Harun@LAPTOP-IDJ190NR MINGW64 ~/NLP/chatbot (master)
$ git branch -M main

Harun@LAPTOP-IDJ190NR MINGW64 ~/NLP/chatbot (main)
$ git remote add origin https://github.com/impactfaith/PizzaRestaurantChatbot.git

Harun@LAPTOP-IDJ190NR MINGW64 ~/NLP/chatbot (main)
$ git push -u origin main
Logon failed, use ctrl+c to cancel basic credential prompt.
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 248 bytes | 82.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/impactfaith/PizzaRestaurantChatbot.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.

Harun@LAPTOP-IDJ190NR MINGW64 ~/NLP/chatbot (main)
$ |
```



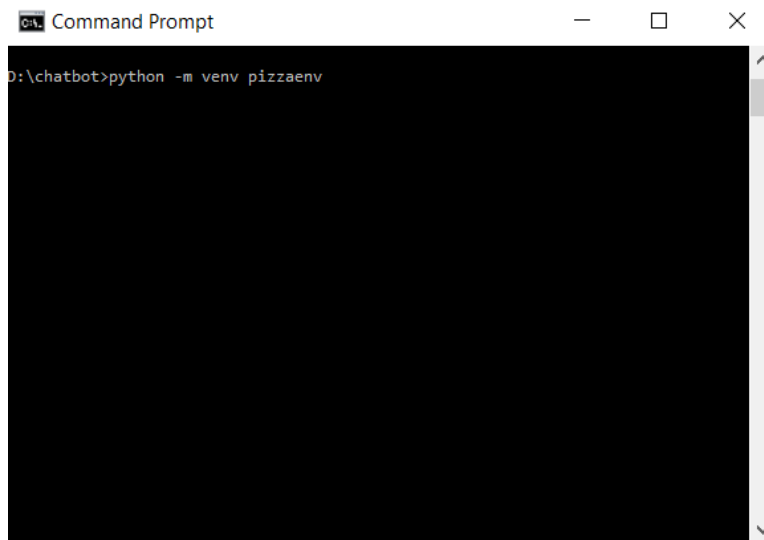
¹ <https://github.com/impactfaith/PizzaRestaurantChatbot>

b) Setting up and activating a deployment virtual environment

First step was to create an isolated python environment mainly to prevent problems like dependency issues. All packages or modules required for this project would be installed and available only in this environment.

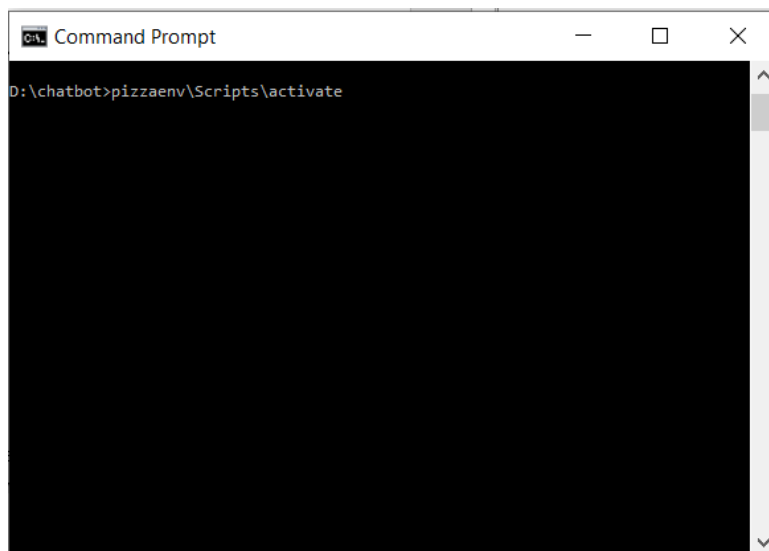
Using an anaconda CMD shell, a python environment called “pizzaenv” was created and activated using the following commands:

Create Environment (***python -m venv pizzaenv***)



```
Command Prompt
D:\chatbot>python -m venv pizzaenv
```

Activate the Environment (***pizzaenv\Scripts\activate***)



```
Command Prompt
D:\chatbot>pizzaenv\Scripts\activate
```

c) Setting up project directory structure

After setting up GIT and the python environment, for proper organization of the several files and resources used in this project, a directory structure was created as shown below

```
Folder PATH structure
C:\Users\Harun\NLP\chatbot:.
|   README.md
|   NLP_Chatbot.docx
+---dataset
|
+---images
|
+---model_artifacts
|
```

Inside the structure, there is the main root of the directory which will contain resources related to Git and this document you are currently reading, a folder to store datasets, folder for images used in the live view and a folder that stores the artifacts of the prebuilt model.

i. Dataset

Inside the dataset folder are 2 files, the “**intents.json**” and “**response.csv**”.

The “**intents.json**” is the dataset written by me which was used to train the model. Since it is a simple chatbot, there was no need to download massive datasets. It is a bunch of messages that the user is likely to type in and mapping them to a group of appropriate responses. The tag on each dictionary in the file indicates the group that each message belongs too. With this data we will train a neural network to take a sentence of words and classify it as one of the tags in our file. Then we can simply take a response from those groups and display that to the user. The more tags, responses, and patterns you provide to the chatbot the better and more complex it will be. A snippet from the json file is shown below

```
{
  "intents": [
    {
      "tag": "start_conversation",
      "patterns": ["Hi there", "Is anyone there?", "Hey", "Hola", "Hello", "Good day", "Hi", "Anyone home?", "Whats cooking", "I am hungry"],
      "responses": ["Hello, I'm sure you love pizza", "Happy to have you here", "Good to see you again", "Hi there, how can I help?", "Good to see you again", "Ready to start?", "Hello, I'm sure you love pizza", "Happy to have you here", "Good to see you again", "Hi there, how can I help?", "Good to see you again", "Ready to start?"],
      "context": [""]
    },
    {
      "tag": "what_are_you",
      "patterns": ["What is your name?", "what are you?", "who are you?", "your name pls?", "identify yourself"],
      "responses": ["Hi, I'm ReadyPizza", "I'm ReadyPizza ", "Call me ReadyPizza", "My friends call me ReadyPizza"],
      "context": [""]
    },
    {
      "tag": "end_conversation",
      "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye", "Till next time"],
      "responses": ["Have a lovely meal!", "Have a nice time eating", "Enjoy your meal!", "Bon appetit!", "eat hearty!", "good appetite."],
      "context": [""]
    },
    {
      "tag": "thanks",
      "patterns": ["Thanks", "Thank you", "That's helpful", "Awesome, thanks", "Thanks for helping me", "Thanks for your time", "That's useful", "Super"],
      "responses": ["Happy to help!", "Any time!", "My pleasure", "Do enjoy your meal", "Have fun eating"],
      "context": [""]
    }
  ]
}
```

The “**response.csv**” is a file generated during modelling stage which contains predefined responses for each tag in the dataset from which the model can choose to respond to a particular question.

	response	labels
1	Hello, I'm sure you love pizza	6
2	Happy to have you here	6
3	Good to see you again	6
4	Hi there, how can I help?	6
5	Good to see you again	6
6	Ready pizza at your service	6
7	Hi, I'm ReadyPizza	9
8	I'm ReadyPizza	9
9	Call me ReadyPizza	9
10	My friends call me ReadyPizza	9
11	Have a lovely meal!	3
12	Have a nice time eating	3
13	Enjoy your meal!	3
14	Bon appetit!	3
15	eat hearty!	3
16	good appetite.	3
17	Happy to help!	7
18	Any time!	7
19	My pleasure	7
20	Do enjoy your meal	7
21	Have fun eating	7
22	Sorry, kindly rephrase the ques	1
23	Sorry, can't understand you	1
24	Please give me more informati	1
25	Not sure I understand	1
26	Please give me more details	1
27		

ii. Images

The images folder contains all the images used in the final hosted chatbot web application to give a familiar feeling that the user is interacting with a pizza ordering chatbot.

iii. Model Artifacts

This folder contains the artifacts generated from the modelling/training stage of the project. They are 4 files that were generated and stored in this folder. Since this project is not about the modelling stage, I will not discuss the details here.

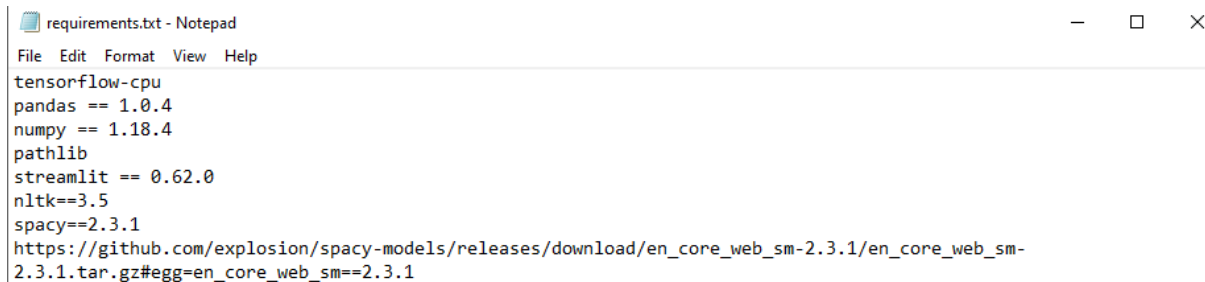
As shown below is the directory structure after populating the folders with their appropriate files.

```
C:\Users\Harun\NLP\chatbot: .
|   README.md
|   NLP_Chatbot.docx
|
+---dataset
|   intents.json
|   response.csv
|
+---images
|   pizzabox.png
|   readypizzalogo.png
|   readypizzadispatch.png
|
+---model_artifacts
|   model-v1.h5
|   tokenizer_t.pkl
|   tokens.pkl
|   vocab.pk
```

d) Scripting

i. Requirements Text file

Next step is to create the various scripts required to deploy, but first up some required packages for the project must be installed. For this a “**requirements.txt**” file was created and placed in the root of the directory, which lists the various libraries and their corresponding versions which will then be installed as a batch in the virtual environment created earlier.



```
requirements.txt - Notepad
File Edit Format View Help
tensorflow-cpu
pandas == 1.0.4
numpy == 1.18.4
pathlib
streamlit == 0.62.0
nltk==3.5
spacy==2.3.1
https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.3.1/en_core_web_sm-2.3.1.tar.gz#egg=en_core_web_sm==2.3.1
```

To install the requirements file:

- Create requirements.txt file with required libraries and versions in the root of the project
- Activate the environment with the command “***pizzaenv\Scripts\activate***”
- Install the file and all its included packages with the command “***pip install -r requirements.txt***”
- is to add the pre-processor python file which was used to pre-process the dataset. It essentially

ii. Pre-processor Python file

Next is to add the pre-processor python file which was used to pre-process the dataset. It essentially contains functions to tokenize, remove stop words and encode the dataset. The file should be added to the root of the project.

```

preprocessor.py - Notepad
File Edit Format View Help
import pandas as pd
import numpy as np
import string
import re
import json

import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import load_model

lemmatizer = WordNetLemmatizer()

def tokenizer(entry):
    tokens = entry.split()
    re_punc = re.compile('%s' % re.escape(string.punctuation))
    tokens = [re_punc.sub('', w) for w in tokens]
    tokens = [word for word in tokens if word.isalpha()]
    tokens = [lemmatizer.lemmatize(w.lower()) for w in tokens]
    # stop_words = set(stopwords.words('english'))
    # tokens = [w for w in tokens if not w in stop_words]
    tokens = [word.lower() for word in tokens if len(word) > 1]
    return tokens

def remove_stop_words_for_input(tokenizer,df,feature):
    doc_without_stopwords = []
    entry = df[feature][0]
    tokens = tokenizer(entry)
    doc_without_stopwords.append(' '.join(tokens))
    df[feature] = doc_without_stopwords
    return df

def encode_input_text(tokenizer_t,df,feature):
    t = tokenizer_t
    entry = entry = [df[feature][0]]
    encoded = t.texts_to_sequences(entry)
    padded = pad_sequences(encoded, maxlen=16, padding='post')
    return padded

```

iii. App Python file

The starting point of the program is the **app.py** which is created and added to the project root. Lines 1–8 import the necessary libraries. Using “**Pathlib**” module which can deal with absolute as well as relative paths, you can specify the frontend images and the model artifacts paths. This makes it possible to run the app on any kind of operating system without any directory or path issues.

We load the images using the image script in the library, load the model artifacts using “**joblib**”, and the model using the “**load_model**” function from the “**tensorflow.keras.models**” script. Other chatbot related functions are defined in this file for instance functions that gets the model or gets the chatbot response.


```

app.py - Notepad
File Edit Format View Help
import numpy as np
import pandas as pd
import preprocessor as p
from tensorflow.keras.models import load_model
import joblib
from pathlib import Path
from PIL import Image
import streamlit as st

#paths
img_path = Path.joinpath(Path.cwd(), 'images')
artifacts_path = Path.joinpath(Path.cwd(), 'model_artifacts')
datasets_path = Path.joinpath(Path.cwd(), 'dataset')

#load images
center = Image.open(Path.joinpath(img_path, 'readypizzalogo.png'))
second_image = Image.open(Path.joinpath(img_path, 'readypizzadispatch.png'))
third_image = Image.open(Path.joinpath(img_path, 'pizzabox.png'))

#load artifacts
model = load_model(Path.joinpath(artifacts_path, 'model-v1.h5'))
tokenizer_t = joblib.load(Path.joinpath(artifacts_path, 'tokenizer_t.pkl'))
vocab = joblib.load(Path.joinpath(artifacts_path, 'vocab.pkl'))

df2 = pd.read_csv(Path.joinpath(datasets_path, 'response.csv'))

```

The directory structure after adding all the files now look like this:

```

Folder PATH listing
C:\Users\Harun\NLP\chatbot:.
|   README.md
|   NLP_Chatbot.docx
|   requirements.txt
|   preprocessor.py
|   app.py
+---dataset
|   intents.json
|   response.csv
+---images
|   pizzabox.png
|   readypizzalogo.png
|   readypizzadispatch.png
+---model_artifacts
|   model-v1.h5
|   tokenizer_t.pkl
|   tokens.pkl
|   vocab.pk

```

e) Deploying with Streamlit

Streamlit was one of the packages installed with the requirements.txt, it is an open-source framework which offers data scientists and machine learning engineers an easy way to turn data scripts into sharable web app in minutes.²

To run the demo, in the activated virtual environment run the command: ***“streamlit run app.py”***, The command tasks streamlit to run the app.py file which is the starting point of the demo.

² <https://www.streamlit.io/>

```
Anaconda Prompt (anaconda3) - streamlit run app.py
The system cannot find the path specified.

(base) D:\chatbot>python -m venv pizzaenv

(base) D:\chatbot>pizzaenv\Scripts\activate

(pizzaenv) (base) D:\chatbot>streamlit run app.py

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.1.103:8501

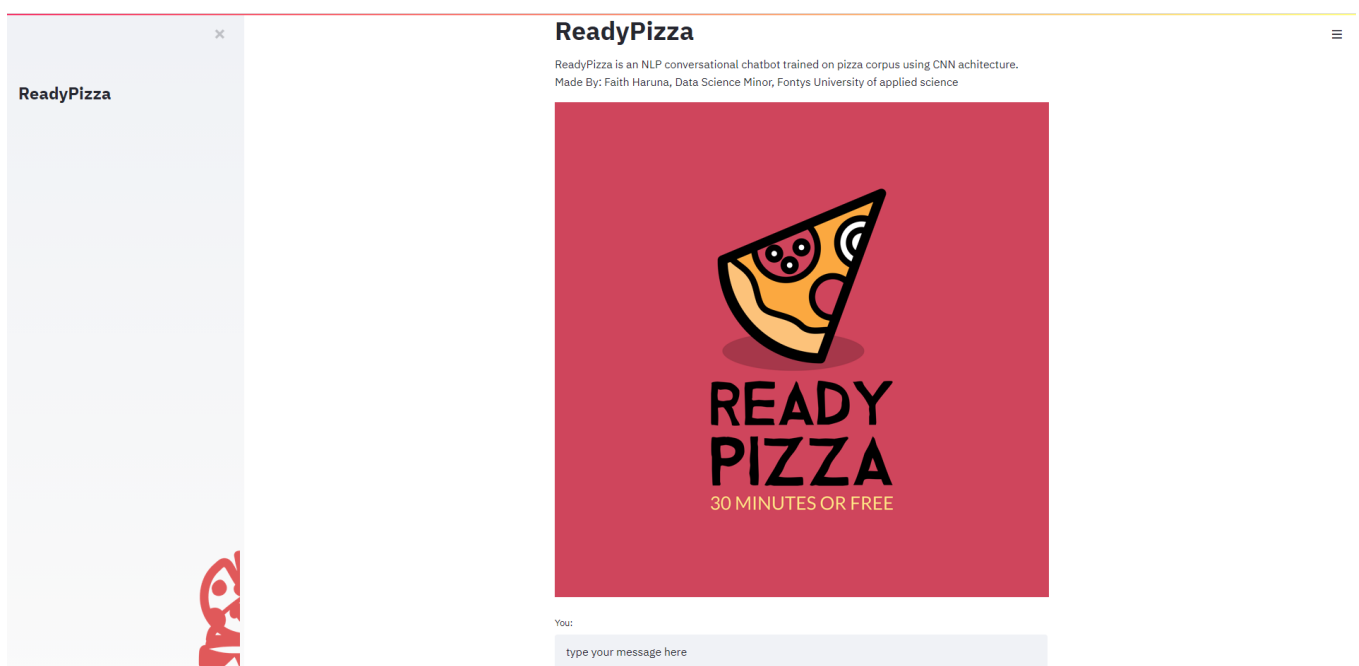
2020-12-17 18:10:06.873020: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not
creating XLA devices, tf_xla_enable_xla_devices not set
2020-12-17 18:10:06.876237: I tensorflow/core/platform/cpu_feature_guard.cc:142]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (on
eDNN) to use the following CPU instructions in performance-critical operations:
AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate comp
iler flags.
2020-12-17 18:10:11.700908: I tensorflow/compiler/mlir/mlir_graph_optimization_p
ass.cc:116] None of the MLIR optimization passes are enabled (registered 2)
Model was constructed with shape (None, 9) for input KerasTensor(type_spec=Tensor
Spec(shape=(None, 9), dtype=tf.float32, name='embedding_1_input'), name='embedd
```

3 Results

After running the app.py script, a web application containing the chatbot and image resources shows up on your screen. Streamlit automatically hosts the application serves a local web application of the chatbot in your localhost and even provide a public iP address to view the live chatbot publicly. In the case of this demo, the live web application can be viewed on <http://192.168.1.103:8501>.

It is important to note that the iP address is dynamic, so the next time it may not be the same iP address that is used to view the app.

The associated resources and files are already added on github
(<https://github.com/impactfaith/PizzaRestaurantChatbot>)



4 Conclusion

This is a simple chatbot deployed with an already trained model using my dataset. To improve on the chatbot, more data should be included to train the model to widen the intelligence of the chatbot. Also, one can skip using streamlit and actually use the technique in a real world web application for a pizza ordering website which has its own dedicated server.

5 References

- A Tennis Chatbot built with Convolutional Neural Networks by **Bamigbade Opeyemi**³
- Python Chat Bot Tutorial - Chatbot with Deep Learning by **Tech with Tim**⁴

³ <https://heartbeat.fritz.ai/building-a-conversational-chatbot-with-nltk-and-tensorflow-part-2-c67b67d8ebb>

⁴ <https://www.techwithtim.net/tutorials/ai-chatbot/part-1/>