# ISYS30221 Artificial Intelligence 2021-22

## Coursework Documentation

## 1- About this submission

|  |  |
| --- | --- |
| Student Name | Elroy D’Souza |
| Student ID | N0858835 |
| Chatbot Topic | European Countries Chatbot |
| Tasks implemented in this submission (a,b,c,or d) | **A**  Implemented rule-based conversation through use of AIML.  Implemented similarity-based conversation using bag-of-words model, TF-IDF, and cosine similarity.  Extra functionality: Use of API to answer questions about currency, flags, etc.  **B**  Implemented first order logic with knowledgebase and inference engine using NLTK library.  Extra functionality: Implemented fuzzy logic inference system.  **C**  Implemented local image classification through use of a convolutional neural network model.  Extra functionality: Implemented multi-object detection in images and implemented manual hyper-parameter tuning.  **D**  Implemented cloud-based image classification through a model developed on Azure cloud with custom vision. Azure image analysis also implemented.  Extra functionality: Implemented facial recognition using an Azure AI service in the computer vision group. |
| Files inventory (excluding this file) | EuropeanCountriesChatBot.py  EuropeanCountriesChatBot.xml  EuropeanQA.csv  kb.csv  FuzzyRules.txt  test\_data folder containing image classification test images  test\_data/mo\_test\_data folder containing multi-object test images  test\_data/face\_test\_data folder containing face analysis test images  EuropeanWondersCNN.py  EuropeanWondersModel.h5 |
| Demo video URL | Task A+B: <https://web.microsoftstream.com/video/761a39cc-6c3c-40e0-a9c2-14717dc17d5e>  Task C+D: <https://web.microsoftstream.com/video/59e74bf2-e5cc-4632-8cd9-dc6559f41c66> |
| Checklist | I will submit this file separately (without compression) into DropBox  All other files are zipped and will be submitted into DropBox  The demo video is recorded as instructed, and the sharing link is inserted above  I have made sure that the demo video is shared according to the instructions, so that I allowed everybody in the university to view it.  All the sections below are populated accordingly. |

## 2- Design notes (shrink/grow as needed, add images where applicable)

|  |  |
| --- | --- |
| General explanations of the system and its goals | The European countries chatbot that has been developed answers questions about countries within the continent Europe. The questions range from displaying a specified countries currency to showing a flag of a certain country. These questions are part of a set of questions that use a REST API, alongside the AIML file, to grab the correct answer. There are other questions, such as asking the chatbot the country it was created in and storing where the user is from, using only the AIML file. The chatbot supports similarity-based conversation too, where if the user enters a question which contains keywords of questions stored within the EuropeanQA.csv file, the closest answer is outputted.  The chatbot system allows users to store data of regions, countries, constituent countries and capitals using a knowledgebase. There is also a fuzzy inference system implemented that allows users to check the fluency of a certain language based on speech and writing skills.  The chatbot has image classification and analysis features to allow detection and analysis of selected European Wonders (Eiffel Tower, Santorini, Stonehenge, Blue Grotto) through local and cloud-based image classification. Facial detection of European leaders (UK, France, Italy, Greece) is also possible through cloud-based facial analysis. There is also a multi-object detection feature that utilises a pre-trained model created by Google to allow detection of main objects within a selected image.  Goals of the system:   * Allow users to input exact questions (with variance, using the <srai> tag in AIML) which will be searched within the AIML file and output a response, utilising an API if necessary. If API is used, handle JSON data within python to output answer. * Allow users to input questions that do not have a match within the AIML file; these questions should be searched within the EuropeanQA.csv file with the closest match being outputted if the cosine-similarity score is above the threshold. * Allow users to utilise the knowledgebase to store data about regions, countries, constituent countries, and capital cities using NTLK’s first order logic syntax. * Allow users to verify data within the knowledgebase through contradiction checking using the rules set within the knowledgebase and NLTK’s resolution algorithm. * Allow users to check their language fluency through entering their speaking and writing abilities on a scale of 1-10. This will use a fuzzy inference system taking in a set of rules defined and performing Mamdani inference to output a result in which will output a specific phrase to state the level of language fluency. * Allow the user to ask for a ‘tip’, which will output a random tip to aid the user to ask suitable questions relating to European countries. * Allow users to detect a European Wonder’s name, from selection, through local and cloud-based image classification and allow analysis of Wonder through cloud-based image analysis. * Allow users to detect European leaders, from selection, through cloud-based facial analysis. * Allow users to detect multiple object names, from selected image, through a pre-trained model.   An example of questions stored in AIML that do not utilise the API:  An example of questions stored in AIML that utilise the API:  The tip functionality uses a <random> tag within the AIML file:  An example of the JSON data stored within the API. |
| The system requirements, i.e., the list of what the system should do/have from a user’s perspective | System requirements from the user’s perspective:   * Display a welcome message which highlights the purpose of the chatbot alongside a prompt for the user to type ‘tip’ if stuck with questions. * Chatbot should reply to a greeting from the user with a randomly selected greeting in the AIML. * Have generic conversation, such as asking how the user is doing, etc. * Chatbot can store the country that the user states they are from and retrieve that name when the user asks. * Display the main currency used within a chosen country. * Display what countries border the chosen country. * Display the country code of a chosen country. * Display the capital city of a chosen country. * Display the subregion of a chosen country. * Display the flag of a chosen country in default image viewer. * Output the answer to questions within the EuropeanQA.csv if user input matches keywords in stored questions. * Informs user with a message if the chatbot cannot answer a question. * Allows user to store data in memory using NLTK’s first order logic syntax, through the command ‘I know that X is Y’. Where Y is either capital/country/constituent country/region and X is the user’s input, e.g. ‘I know that London is capital’. Contradiction checker will check if the input is valid. * Allows user to check the knowledgebase using the command ‘Check that X is Y”. This will check the knowledgebase for if statement is true. * Allow user to check their fluency of languages through entering their speaking and writing skills. * Allow user to detect a European Wonder name, through selecting image, with local and cloud-based classification result given. * Allow user to analyse European Wonder image through selecting image. * Allow user to detect European leader name/s of leaders within a selected image. * Allow user to detect multiple object names within selected image. |
| The employed AI techniques, and the explanation of program codes and the supplied files. | AI techniques used by both task A and B:  **Task A:**  This task required the chatbot to be able to perform rule-based conversation. This was done using the AIML module to provide pattern matching in such a way to resemble human conversation. AIML stands for ‘Artificial Intelligence Markup Language’ and utilises categories, templates, and patterns to define responses to the input it is given. AIML uses tags to identify text that is interpreted by an AIML interpreter, such as ‘<pattern>’, ‘<random>’, or ‘<star>’. This markup language was implemented into the chatbot within a separate file called EuropeanCountriesChatBot.xml which stored answers to certain inputs. For example, the AIML file utilised the ‘<set>’ tag to store the country the user is from through the command ‘I am from X’, with X being the name of the country. Where the user can then continue by asking what country they are from, and the AIML tag ‘<get>’ is used to find the stored country value.  Task A also required the chatbot to handle similarity-based conversation. Bag-of-words, TF-IDF, and cosine similarity were implemented to score the user input with a set of questions. If the user asked the chatbot a question that cannot be matched within the AIML file, then the three techniques mentioned above would be used to try find a likely match to the question. The first step to creating this system is by reading all the questions and answers stored within the EuropeanQA.csv file and splitting them up into two different lists. Then when a question is asked, for example, ‘what is the largest country in Europe by landmass?’ it is converted into a matrix of TF-IDF features through the ‘TfidfVectorizer’ function imported by the sklearn library. This function combines the features from ‘CountVectorizer’ and ‘TfidfTransformer’. ‘CountVectorizer’ implements occurrence counting, which creates a bag-of-words model of each sentence, whilst ‘TfidfTransformer’ implements weighting to each bag-of-words model based on the word occurrence and how common they are. The ‘TfidfVectorizer’ is applied to the stored questions and user input, with the ‘stop\_words=”english”’ parameter being passed through to get rid of non-essential words such as ‘is’ and ‘the’. Cosine similarity is then implemented through the function ‘cosine\_similarity’ on the TF-IDF matrices of stored questions and the TF-IDF matrix of the user input. Cosine similarity rates each question in the stored questions with the user input and gives them all a similarity score. The question that matched the closest can then be picked and the answer to that question outputted to the user. A threshold of 50% has been set so that if there is no stored question that matches more than 50% of the user input, the user is notified that their question could not be understood.  **Task B:**  This task utilises the NLTK library to create an inference engine using a first order logic knowledgebase. Two patterns are stored within the AIML file, ‘I know that X is Y’ and ‘Check that X is Y’, X being an object and Y being a subject. These patterns allow the user to input statements to save within memory (not in file) and check statements that are stored within the knowledgebase. Certain rules should be set within the knowledgebase file (kb.csv) to allow the check statement to determine whether a statement is false or unknown. These rules need to be written in NLTK’s first order logic syntax, e.g., ‘Capital (x) -> -Country(x)’. The values that are stored within the kb.csv also need to follow the same syntax rules, e.g., ‘Region (NorthEurope)’ or ‘Capital (Paris)’. Contradictions are checked using the function ‘ResolutionProver’ imported from the NLTK library. A negative version of the statement is set to a new variable which is then put through the ‘ResolutionProver’ to check if the statement contradicts with any rules, and if so, the user is notified there is a contradiction. This is also done at the start of the whole python program to make sure no contradictions are found within the knowledgebase, if any are found the program is terminated.  For this task, fuzzy logic, was also implemented to meet criteria for extra functionality. This was implemented into a fluency checker system that the chatbot allows the user to interact with. The user can start it up through the command ‘fluency checker’ or similar, which is matched with a pattern in the AIML file. The code then asks the user for their speech and writing skills both rated out of 10. These inputs are then taken in and placed on the fuzzy variables scale defined in the code. The fuzzy sets utilise the simpful library to create boundaries to split up the variables into low, average, and high. These variables are then placed within an output variable which utilises certain rules that have been set within text file FuzzyRules.txt. Mamdani inference is then performed on this output variable based on the universe of discourse of the variable. The result can then be used to output a certain phrase based on a range that has been set. These phrases then inform the user how fluent they are on the language chosen.  AI techniques used by both task C and D:  **Task C:**  This task required the chatbot to be able to utilise a locally trained model to classify an image given by the user. This task therefore uses a large image dataset for each chosen class and a convolutional neural network (CNN) model. This would be triggered through an AIML pattern with the user entering the command ‘what is this European Wonder’ or similar. Upon entering the command, a file dialog will open that would allow the user to select an image of a European Wonder of their choosing. There are only four European Wonder classes, those being the Eiffel Tower, Santorini, Stonehenge, and the Blue Grotto. After the user has selected an image, a prompt asking whether the user would like to analyse or identify the image is given. Entering ‘analyse’ utilises cloud-based image analysis which will be expanded upon within the task D section. Entering ‘identify’ utilises both local and cloud-based image classification to detect the name of the European Wonder. As task C is focussed on local image classification, the cloud-based classification will be expanded upon within the task D section. The code utilises the CNN model that has been created through loading the .h5 file saved from the model. This will then detect which class it predicts the image falls under and outputs the name alongside a percentage of likeliness.  The code within the CNN model will be expanded upon within the code explanation section below, however the performance will be reported here. As one of the extra functionalities for Task C, hyperparameter tuning optimisations were made to improve the performance of the CNN model. The model used as a starting point was taken from the TensorFlow image classification tutorial. Hyperparameter tuning was mainly done manually as Hyperband Tuner was attempted through testing for dropout, the first dense layer, and learning rate, however the best validation accuracy would always be a lot worse with these tests as opposed to the original model. This led to manually tuning the model to see improvements in validation accuracy. The reason validation accuracy was used as the main measurement is because this percentage shows how the model is expected to perform on new data, which is what is being tested in task C.  The first test that was performed was testing if having a dropout layer after the final max pooling layer improved performance. The validation accuracy difference was negligible so further tests were performed at any future model changes, all of which proving there was little difference in performance, leading to not including this layer. Another test was on adding another convolutional layer to increase complexity of the model. This instantly saw an increase in the validation accuracy and reduced overfitting in the model, this may be due to the training set being larger than the original models’ network. One issue that was then seen when plotting the model’s accuracy and loss was an increase in divergent behaviour within the loss function. Tuning the optimisers learning rate to make it a lot lower, from 0.01 to 0.0001 made the models line a lot smoother, reducing divergence. This in turn meant the number of epochs would need to be increased, from 10 to 50, as the model was no longer reaching its minimum point. The only other test that was performed was changing the first dense layer’s units between the range of 64 and 256, from 128. However, increasing the units seemed to increase overfitting, whilst decreasing units lowered accuracy a lot. This all led to the final model that was created which had a best validation accuracy of 95.71% and training accuracy of 98.21%. These are seen as very high accuracies as it can be stated the model has a ~95% chance to predict the European Wonder’s class correctly. The training and validation data accuracy and loss graph of the final model is shown below.  Looking at the graph above, there is some overfitting that can be seen between the validation and training data; however, it is a lot less after the tuning of parameters. Even through the learning rate of the model was lowered a lot, the model can be seen to have a high learning rate with the training and validation accuracy increasing by ~40% within the few starting epochs.  Another extra functionality was also implemented for task C, that being multi-object detection within images. This functionality can be triggered through the command ‘what objects are in this image’ or similar. This will cause the chatbot to load the pre-trained region-based convolutional neural network (RCNN) model created by Google. This uses an image feature extractor, Inception Resnet V2, to read objects from the image. These objects are then tested on the RCNN model to identify the closest match out of 600 pre-set classes. Once the model has identified the predicted objects, all object names with a prediction score above 9% are outputted alongside the percentage of confidence.  **Task D:**  This task required the chatbot to identify selected images through a cloud-based image classification service. This uses Azure’s custom vision resource, where the image dataset is uploaded onto the custom vision website to be trained using a pre-built model. As the image dataset uploaded only consists of European Wonders, the domain can be set as ‘landmarks’ to improve performance of the model. Training the dataset with the default model on the first iteration that was run produced the performance shown below.  This performance is perfect getting 100% across all categories, meaning the model would be very likely to classify images correctly. The user can access the cloud-based image classification result through typing the command ‘what is this European Wonder’, selecting an image, and entering ‘identify’ to the prompt asking whether the user wants to analyse or identify the image. This will output the result given by Azure’s model alongside the task C CNN models result, which should be the same if both models work as intended.  Alongside the custom vision resource implementation, Azure’s computer vision resource was also utilised as a side-feature to analyse images. When the user is prompted asking whether they want to analyse or identify an image, if ‘analyse’ is entered, Azure attempts to describe the contents of the image. This could consist of detecting specific objects and/or generating text-based summaries. The computer vision resource can do this through utilising pre-trained machine learning models that are able to extract information and analyse them.  For this task, facial recognition was also implemented as an extra functionality. This was done using a cloud-based facial cognitive service that utilises computer vision solutions. Facial recognition works through the Azure service analysing photos of faces given to it. All facial features of each face are then stored onto Azure and given a key. This key can then be matched with a future key that is generated when testing an image that contains a face within it, and if there is a match then the specific face has been detected and can be output. This facial recognition feature can be triggered through the command ‘who is this European leader’ or similar. This allows the user to select an image of their choosing. Error checking has been implemented for if no faces are recognised, and if there are none of the European leader faces recognised. However, if there is one of the four European leader (UK, France, Italy, Greece) faces spotted within an image, the chatbot outputs every leader spotted within the photo.  Explanation of program codes and supplied files:  **EuropeanCountriesChatBot.py:**  This file contains all the python code that provides functionality to the chatbot to complete all four tasks.  At the start of this file all the imports are declared. They are all checked with pypi.org to ensure they are safe libraries to use. Following are the reasons why certain imports were used:   * Import json: standard library used to handle JSON data. * Import CSV: standard library used to read CSV file. * Import os: standard library used to read files. * Import tkinter: standard GUI library used to launch file dialog. * Import regex: used to return the string matched by the regular expression. * Import PIL: used to display image on default image viewer. * Import requests: used to grab JSON data from API URL. * Import AIML: reads AIML file. * Import sklearn: provides functions ‘TfidfVectorizer’ and ‘cosine\_similarity’. * Import pandas: allows a data frame to be created from kb.csv. * Import simpful: provides fuzzy logic inference engine. * Import NLTK: provides first order logic inference engine. * Import tensorflow: provides local image classification support. * Import tensorflow-hub: provides multi-object detection support. * Import numpy: provides functions for numerical arrays. * Import azure computer vision: provides cloud-based image analysis support. * Import azure face: provides cloud-based facial recognition support. * Import azure custom vision: provides cloud-based image classification support.   The chatbot reads the knowledgebase and performs an integrity check through checking contradictions. If a contradiction is found the program is terminated.  The checkEurope(country) function checks to see if the country is located within Europe, and if it isn’t an error message is displayed.  The kbFormatting(object, subject) function is used to format the kb inputs so that it does not cause errors within the knowledgebase.  The trainFaces(groupID) function is used to train faces of European leaders, however this function has no use in the current system as this only needs to be run once. The addPerson and registerPhotos functions are extensions to the trainFaces function.  The detectorLoad(moduleHandle) function is used to load the RCNN model supplied by Google, this can only be run once per run of the application.  The AIML file is read and depending on the user input a certain value is stored under variable name ‘cmd’. This allows the program to jump to a certain function that will perform the functionality that is required. For example, if the user asks, ‘what is the currency in France’, the AIML file outputs 2. This will trigger the if statement asking if ‘cmd’ is equal to 2 which will in turn grab the currency of the countries name ‘France’ using the API. If there is no command found, the chatbot will proceed to print the answer given by the AIML file.  Following is a list of every command and what it does:  0: statement break is used to exit the while loop, in turn terminating the chatbot.  2: currency of the country specified is outputted.  3: name of all countries that border specified country is outputted.  4: country code of the country specified is outputted.  5: capital city of the country specified is outputted.  6: subregion of the country specified is outputted.  7: flag of the country specified is displayed on default image viewer.  50: allows user to store value within memory, if it does not contradict with anything within the knowledgebase  51: checks the validity of a statement, if the statement is true then chatbot will output correct. Else if the statement is disproved, through contradiction checking of the knowledgebase, then the user is informed that the statement is false. However, if the knowledgebase cannot find it is a contradiction, then the user is informed it isn’t sure if it is true or false.  52: this is the fuzzy inference system that allows the user to check language fluency through entering their speaking and writing skills, ranked out of 10. There is more information on how this system works on page 5 of this document.  53: the image analysis or classification of a user selected European Wonder. Selecting ‘analyse’ will use Azure computer vision to describe the image, whilst selecting ‘identify’ will use both Azure custom vision and CNN model to classify the image.  54: this is the facial recognition feature that uses Azure computer vision to detect and output the name of the European leader within the selected photo.  55: this is the multi-object detection feature that utilises a TensorFlow RCNN model created by Google to detect and output the names of objects within a selected photo.  99: this command is used when there are no matches to the AIML file, it contains the code for similarity-based conversation functionality. There is more information on how the similarity-based conversation works on page 4-5. If the threshold isn’t reached for cosine similarity score, then a message informing the user their question could not be understood is outputted to the user.  **EuropeanCountriesChatBot.xml:**  This XML file contains all the patterns structured in AIML markup language. It is the primary function that allows the chatbot to conversate with the user allowing for inputs and outputs. The ‘<srai>’ tag is used to allow alternative ways of asking a question. Following are different features of the AIML file:   * A greeting selects a random phrase to greet the user with, using the ‘<random>’ and ‘<li>’ tags. The ‘tip’ command entered by the user also utilises the random tag. * Questions that are not using the API are listed with answers using the standard ‘<pattern>’ and ‘<template>’ tags. There is also a ‘<set>’ and ‘<get>’ tag used to store the country the user is from and grab that country later. * Questions that utilise the API are listed with their respective command code. * A default reaction using the ‘<star>’ functionality that will grab any input that does not match any other statement within this AIML file. * Knowledgebase statements are listed with their respective command code. * Fuzzy inference system phrase is listed with its respective command code. * European Wonder classification through local and cloud-based models command code. * European leader facial detection command code. * Multi-object command code.   **EuropeanQA.csv:**  This file is used to store extra questions and answers for the similarity-based conversation system. These questions are vaguer one’s that the user may ask, as opposed to the direct ones about specific countries that require the API.  **kb.csv:**  This file is used to store the rules required for the first order logic inference system. The first four lines are specific rules on the subject’s used, whilst all other lines are storing objects into these subjects.  **FuzzyRules.txt:**  This file contains all the rules that are used by the fuzzy inference system.  **EuropeanWondersCNN.py**  This file contains all the python code required to build a convolutional neural network (CNN) model for the four European Wonder classes. This is code required for task C.  At the start of the file there are imports declared. All these imports have been checked with pypi.org to ensure they are safe to use. Following are the imports with what they were used for:   * Import os: standard library used to read files. * Import matplotlib: provides functionality for plotting values onto a viewable graph. * Import tensorflow: provides local image classification support. * Import keras: implementation of the keras API (auto installed with tensorflow).   Constant variables batch size, image height, and image width are declared for easy parameter value changes.  Images are loaded from directory and placed into separate training and validation datasets with a validation split of 0.2. These images are automatically given specific labels (0, 1, 2, 3) dependant on which subdirectory they are within. A random seed is set to shuffle the data and the image size and batch size are set to the constant values set beforehand. Images are kept RGB as ‘color\_mode’ parameter is not changed, meaning it will need 3 channels.  Training and validation data is saved to cache for faster run times. This works through prefetching data which creates an input pipeline to read data whilst other data is being trained, reducing training and extraction time.  Data augmentation is defined under a variable to increase dataset size with random changes, such as horizontal flip, zoom up to 10%, and contrast change up to 10%. These are all extra layers that will be added to the model before pre-processing.  Sequential model is created, with the augmented data being inputted into the model first before the stack of layers. There are 12 layers that follow:   * Rescaling Layer: pre-processing layer which rescales values to a new range, essentially performing image normalisation, for the purpose of providing a uniform scale before building the model. * 2D Convolutional Layer: convolutional layer where 16 output filters are applied over the 2D input data, activation ‘relu’ is used as it is most suited for CNN’s. * 2D Max Pooling Layer: pooling layer to down-sample the input, reducing its dimensionality. This helps reduce overfitting and computational cost by reducing the number of parameters to learn. * 2D Convolutional Layer: convolutional layer where 32 output filters are applied over the 2D input data, activation ‘relu’ is also used. * 2D Max Pooling Layer: another pooling layer applied for down-sampling. * 2D Convolutional Layer: convolutional layer where 64 output filters are applied over the 2D input data, activation ‘relu’ is also used. * 2D Max Pooling Layer: another pooling layer applied for down-sampling. * 2D Convolutional Layer: convolutional layer where 128 output filters are applied over the 2D input data, activation ‘relu’ is also used. * 2D Max Pooling Layer: final pooling layer applied to down-sample. * Flatten Layer: flatten layer removes all dimensions except for one, essentially making a 1D array of elements. * Dense Layer: first dense layer feeds all outputs from the previous layer to all 128 neurons. * Dense Layer: final dense layer feeds all outputs to the number of classes within the model, which in this case is four for each European Wonder.   The model is configured with various parameters, these being:   * Optimiser set to ‘adam’ with a learning rate of 0.0001. * Loss set to ‘sparse categorical crossentropy’ with ‘from\_logits’ set to true, as to inform that the model is not normalised due to ‘softmax’ activation not being applied to the final layer. * Metrics set to ‘accuracy’.   A summary of the model is printed to view total parameters which helped when hyperparameter tuning.  A checkpoint call-back is defined to save the model when the validation accuracy is at the highest point.  The model is trained with 50 epochs and the checkpoint call-back is set. This data is saved to a variable to be plotted on a graph.  All output values are stored into variables, and these are used to create a graph for tracking training and validation’s accuracy and loss values against each other. This is a method of measuring accuracy and loss to see if there is any over or under fitting within the model.  Below is an example of the output given when this python file is run. The validation accuracy is tracked through the checkpoint call-back, and the model is overwritten every time the highest validation accuracy is seen. This gives the model a higher chance to identify the correct European Wonder more accurately.  **EuropeanWondersModel.h5**  This file contains the model that can be loaded within the EuropeanCountriesChatBot.py file for local image classification of a European Wonder. |

## 3- Conversation log (insert text, screenshots and/or images as required)

## 10 conversation pairs for each task are enough)

|  |
| --- |
| **Task A conversation log:**  Rule-based component using AIML rules, <set> tag used to store user’s country which is later retrieved using the <get> tag. <random> tag is used for greeting and to display a random tip for the user.    Similarity-based component using bag of words model, TF-IDF, and cosine similarity on questions/answers stored in EuropeanQA.csv file.  Rule-based component using AIML rules, python code is used to retrieve and handle JSON data from the API to grab bordering countries to display.    Rule-based component using AIML rules, python code used to retrieve image URL from API. Pillow library is used to display this image in the user’s default image viewer.    Similarity-based component using bag of words model, TF-IDF, and cosine similarity on questions/answers stored in EuropeanQA.csv file.    Rule-based component using AIML rules, python code used to retrieve necessary JSON data of country using the API.  Rule-based component using AIML rules to exact  **Task B conversation log:**  Since the chatbot topic is about European Countries, here is the data stored within the knowledgebase:   * Capitals are not countries. * Capitals are not regions. * Countries are not capitals. * Countries are not regions. * Constituent countries are countries. * North Europe is a region. * East Europe is a region. * South Europe is a region. * West Europe is a region. * United Kingdom is a country. * London is a capital. * France is a country. * Paris is a capital. * Italy is a country. * Rome is a capital. * Estonia is a country. * Tallinn is a capital. * Wales is a constituent country. * England is a constituent country.   The knowledgebase file, kb.csv, is written in NLTK’s first order logic syntax:  When the python program is run, a contradiction check will be made on the knowledgebase, and if any contradictions are found the program is terminated. As seen below, if we store Rome as a region, the program is terminated as Rome is already stored as being a capital, where capitals cannot be regions.      The user has this conversation with the chatbot as an example showing all outcomes. Objects and subject do not need to be capitalised or have proper spacing, as this is all formatted within the code. Adding new values into the knowledgebase will only add it into memory and not the file.  The fuzzy inference system that was implemented as extra functionality for task B. Rules are taken from the FuzzyRules.txt file and used by the Mamdani inference to output a value. This value can then decide which phrase is outputted to the user.  The rules for the fuzzy logic system:  **Task C conversation log:**  The user can type in the command below ‘what is this European Wonder?’ or similar, which will launch a file dialog from the user’s operating system as seen below:  The user can then select the image ‘EiffelTower2’ and the chatbot will give the following response:    The user then must type the word ‘analyse’ or ‘identify’ depending on their choice, however for this task the ‘identify’ option is all that is needed to display the results for task C. The chatbot will then proceed to give the Azure custom vision result alongside the CNN model result, which is the task C result. This result shows the European Wonder class the model predicts with the percentage of likeliness next to it, as seen below:  As you see above, the CNN model correctly identified the European Wonder as the Eiffel Tower, with a high level of confidence.  Below are more examples of user conversations using task C’s CNN model:    With the result seen below, the CNN model has a lot lower confidence percentage. This shows that the model is not very sure if this is the correct image, however, it does still correctly identify it as the Eiffel Tower.          The CNN model can sometimes misidentify the image given, especially if the image is very obscure like the one shown to the left. Azure custom vision correctly identified the close-up of Stonehenge, however the CNN model predicted it being Santorini.                            For one of the extra functionalities for task C, a multi-object detection RCNN model has been integrated into the chatbot. After entering the command ‘what objects are in this image?’ or similar, the message below is seen:  This is put to inform the user the model is currently being loaded, as this process takes some time. Once it is loaded the user is then prompted with a file dialog like below, where they can select an image of their choosing:  Once the image has been chosen, the RCNN model will output all object names over 9% confidence with their percentage confidence next to it, as shown below:    Looking at the image to the left, the chatbot has correctly identified different parts of the image. The reason there are three towers could be due to the three segments of the Eiffel Tower. There are several streetlights on the ground resulting in a few being detected with higher than 9% confidence. There are also several trees surrounding the tower, however they cannot be seen very clearly, hence the low percentage.  Another example of the multi-object detection can be seen below:    The model has identified one pillar of the Stonehenge as a ‘sculpture’ which is a very close prediction. It has correctly identified the person and the clothing they are wearing. The model has also correctly identified the ladder within the image; however, it might have also mistakenly predicted the scaffolding as a ladder too as it predicts several ladders. The tree prediction could be due to the line of trees in the background of the image.    **Task D conversation log:**  The custom vision conversation log to this task can be seen within all core task C images above, however, will be shown below for consistency purposes.  For the user to access cloud-based image classification of chosen images, they must enter the same command as task C ‘what is this European Wonder?’ or similar. Then they will be prompted with the same file dialog as task C, shown below:    The user is then prompted with the same question as shown in task C, ‘would you like to analyse or identify this image?’, upon entering ‘identify’ the chatbot uses the custom vision model created on Azure to classify the image. This classification is output to the user as seen below:      Below are more examples of user conversations using task D's cloud-based classification model:          Azure custom vision is highly accurate and has not misidentified any images given during testing. However, giving it a completely random image does lead it to assigning a random classification it predicts is closest. This can be seen below:      Azure computer vision services has also been implemented through an image analysis feature. This can be accessed through the same command ‘what is this European Wonder?’ or similar. However, upon receiving the question ‘would you like to analyse or identify this image?’ the user must enter ‘analyse’ for the Azure image analysis feature to be used. This feature will read the image and attempt to describe what is happening within the image, giving a percentage of confidence alongside it, a few examples can be seen below:          For the extra functionality for task D, facial recognition using cloud-based services was implemented. After the user inputs the command ‘who is this European Leader?’ or similar, the file dialog opens as shown below:  The user can then continue to select an image and the chatbot will use Azure’s computer vision facial recognition system to detect any European leaders within the photo. The output of the image selected is shown as below:    Azure can detect multiple European leaders, that have been trained, within this photo, leading to both leaders being displayed to the user as shown below. Although all people have recognised faces, only the trained European leader names will be displayed to the user.      The chatbot is also able to identify when a single European leader is spotted within an image.      When there are no trained European leaders spotted within a photo, the following output is given to the user.      The chatbot is also able to identify when there are no faces detected within an image. The following output is given to the user. |