Module `chatbot`

Sub-modules

Module `chatbot.AIMLEngine`

AIML engine module is used to perform the AIML based functionalities of the chat bot.

The patters of the conversation are loaded in from pre-defined in an xml file.

Functions

Function `get_response`

```
def get_response(
    query: str
) -> str
```

Get the response from the AIML agent.

Args ——= **`query`** : User query

Returns: Response from AIML agent

Function `load_aiml`

```
def load_aiml(
    filepath: str
) -> None
```

Loads AIML file into the module.

Args ——= **`filepath`** : Path to AIML file

Module `chatbot.AzureObjectDetectionEngine`

Azure Object Detection Engine used to provide azure object detection services on client side.

Functions

Function `_draw_on_frame`

```
def _draw_on_frame(
    frame: <built-in function array>,
    items: List[Dict[~KT, ~VT]]
) -> <built-in function array>
```

Draws provided inference results on the provided frame.

Args ——= **`frame`** : Frame to draw on

`items` Inference results to draw

Returns: Frame with inference results drawn on it

Function `_get_random_color`

```
def _get_random_color() -> Tuple[int, int, int]
```

Get a random color for the cv2 plots.

Returns: Tuple, with 3 values ranging from 0 to 255

Function `inference_from_file`

```
def inference_from_file(
    image_path: str
) -> None
```

Runs the inference on frame after loading it in from the path provided.

The inference results are displayed to the user via CV2 window

Args —= **`image_path`** : Path to the image on the filesystem

Function `load_credentials`

```
def load_credentials(
    endpoint: str,
    subscription_key: str
) -> None
```

Load in the credentials for Azure Computer Vision Service

Args —= **`endpoint`** : Endpoint

`subscription_key` Subscription key

Module `chatbot.ClassificationEngine`

Classification Engine used to provide local image classification from pre trained CNN.

Functions

Function `classify_from_file`

```
def classify_from_file(
    img_path: str
) -> Tuple[str, float]
```

Loads the image from the filesystem and returns classification results.

The classification network must be loaded in previously via load_model() function

Args —= **`img_path`** : Path to the image on the filesystem

Returns: Classification results as tuple (class_id / class_name , score)

Function `classify_from_image`

```
def classify_from_image(
    img: <built-in function array>
) -> Tuple[str, float]
```

Takes the numpy array as image input and performs classification, returning classification results.

The classification network must be loaded in previously via load_model() function

Args —= **`img`** : CV2 Style numpy image

Returns: Classification results as tuple (class_id / class_name , score)

Function `load_model`

```
def load_model(
    model_filepath: str = '/home/ivica/Coding/uni-chat-bot/chatbot/model.h5',
    input_size: tuple = (224, 224),
    classes: list = []
) -> None
```

Loads image classification neural network into program.

Classes argument can be omitted, in that case classification returns class ID.

Args —= **`model_filepath`** : Filepath to the model

`input_size` Image size for the input layer
`TODO` Expand the tuple size to include also the channels, i.e. (224,224,3)
`classes` List of classes model can detect, if omitted classification returns class ID

Returns:

Module `chatbot.KBEngine`

The KBEngine module is used to provide the logical reasoning capabilities with the help of the NLTK library.

Initial logic is loaded into the chatbot from the Knowledge base txt file.

Functions

Function `load_knowledge_base`

```
def load_knowledge_base(
    filepath: str
) -> None
```

Loads knowledge base from external txt file into the module

Args —= **`filepath`** : Path to the txt KB file

Function `prove_statement`

```
def prove_statement(
    a: str,
    b: str,
    c: str
) -> bool
```

Prove statement using NLTK Inference Resolution Prover.

Format for proving > a(b,c)

Args —= **`a`** : Word

`b` Word
`c` Word

Returns —= Validity of statement

Module `chatbot.QAEngine`

The QAEngine module is used to perform similarity-based question lookup to provide the user with the best possible answer.

The similarity-based functionality is based on a set of pre-defined Q/As in a CSV file. The similarity-based component is based on the bag- of-words model, tf/idf, and cosine similarity.

Functions

Function `_get_real_question_id`

```
def _get_real_question_id(
    question: str,
    confidence_threshold: float = 0.0
) -> Tuple[bool, int]
```

Perform the similarity-based lookup for the real question from our QA list based on the user-entered question.

Similarity based lookup based on bag of words and cosine similarity is used to determine the question the user most likely wanted to ask. User question is appended to the question list and sparse matrix is created and passed to the pandas data frame. Afterwards the cosine similarity is calculated using sklearn, our question is removed from the question list and similarity list (as it's score is always 1.00). Finally, the index with biggest score is returned. Note, in order to exclude useless answers, the confidence threshold is applied.

Args ——= **`question`** : User question to apply similarity-based lookup on

`confidence_threshold` Confidence threshold for cosine-similarity. Used to exclude useless answer

Returns: Validity status, Index of question in _questions list best matching to User question input

Function `get_answer`

```
def get_answer(
    question: str,
    confidence_threshold: float = 0.25
) -> Tuple[bool, str]
```

Interface function used to obtain the answer for the question provided, running similarity-based lookup in the background.

Args ——= **`question`** : User question

`confidence_threshold` Confidence threshold for cosine-similarity. Used to exclude useless answer

Returns ——= Validity status ,answer to user question

Function `load_qa_csv`

```
def load_qa_csv(
    filepath: str
) -> None
```

Function used to load qa csv file into module.

Args ——= **`filepath`** : Path to csv file

Function `load_qa_pair`

```
def load_qa_pair(
    question: str,
    answer: str
) -> None
```

Load the QA pair into QAPair module.

Args ——= **`question`** : Question

`answer` Answer

Function `print_qa_pairs`

```
def print_qa_pairs() -> None
```

Print QA Pairs for debug purposes.

Module `chatbot.TranslateEngine`

Translate engine used to provide azure powered translation functionalities on client side.

Functions

Function `load_credentials`

```
def load_credentials(
    subscription_key: str,
    location: str = 'global',
    endpoint: str = 'https://api.cognitive.microsofttranslator.com'
) -> None
```

Load in credentials for Azure Translator Service

Args —= **`subscription_key`** : Subscription key from Azure

`location` Location for the translator, default global
`endpoint` Endpoint for translation service

Function `translate`

```
def translate(
    input_text: str,
    input_language: str = 'en',
    output_language: str = 'hr'
) -> str
```

Translate the input text to target language

Args —= **`input_text`** : Input text

`input_language` Input language code, default en
`output_language` Output language code, default hr

Returns: Translated text

Module `chatbot.WikiApi`

WikiAPI Module used to interface with python wikipedia module.

Used when user want to retrieve the data from wikipedia on a given topic trough chat bot.

Functions

Function `get_from_wiki`

```
def get_from_wiki(
    topic: str,
    sentences=3
) -> Tuple[bool, str]
```

Get the information from wikipedia on provided topic using python wikipedia module.

Args —= **`topic`** : Topic of interest

`sentences` Number of sentences on the topic

Returns: Validity status, Details about the topic

Module `chatbot.YoloV5ObjectDetectionEngine`

Yolov5 object detection engine based on https://github.com/ultralytics/yolov5 used to provide object recognition capabilities locally.

Functions

Function `_draw_on_frame`

```
def _draw_on_frame(
    frame: <built-in function array>,
    items: List[Dict[~KT, ~VT]]
) -> <built-in function array>
```

Draws provided inference results on the provided frame.

Args —= **`frame`** : Frame to draw on

`items` Inference results to draw

Returns: Frame with inference results drawn on it

Function `_get_random_color`

```
def _get_random_color() -> Tuple[int, int, int]
```

Get a random color for the cv2 plots.

Returns: Tuple, with 3 values ranging from 0 to 255

Function `_inference_frame`

```
def _inference_frame(
    img: <built-in function array>
) -> List[Dict[~KT, ~VT]]
```

Runs the inference from the supplied cv2 frame.

The detection network is must be loaded in via load_network() function before attempting inference

Args —= **`img`** : CV2 Type frame to run the inference on

Returns: Inference results

Function `inference_from_file`

```
def inference_from_file(
    img_path: str
) -> None
```

Runs the inference on frame after loading it in from the path provided.

The inference results are displayed to the user via CV2 window

Args —= **`img_path`** : Path to the image on the filesystem

Function `inference_on_camera`

```
def inference_on_camera(
    camera: str = '/dev/video0'
) -> None
```

Runs the inference from the frames incoming via camera provided.

The inference results are displayed to the user via CV2 window

Args —= **`camera`** : System path to camera, /dev/video* on Linux

Function `load_network`

```
def load_network(
    model_path: str,
    input_width: int = 640,
    conf_threshold: float = 0.25,
    iou_thres: float = 0.45,
    classes: list = []
) -> None
```

Load the Yolov5 neural network for the inference.

Classes parameter can be omitted. If not provided, drawing on the frame will use class numbers instead of class labels.

Args —= **`model_path`** : Path to the yolov5 model on filesystem

`input_width` Input width to the model, default 640
`conf_threshold` Confidence threshold for non-maxima suppression
`iou_thres` IoU threshold for non-maxim suppression
`classes` Array holding list of classes

Classes

Class `ObjectDetection`

```
class ObjectDetection(
    model_path: str,
    input_width: int = 320,
    conf_threshold: float = 0.25,
    iou_thres: float = 0.45
)
```

Initialise the object detecion class.

Args —= **`model_path`** : Path to the yolov5 model on filesystem

`input_width` Input width to the model, default 640
`conf_threshold` Confidence threshold for non-maxima suppression
`iou_thres` IoU threshold for non-maxim suppression

Methods

Method `detect`

```
def detect(
    self,
    input_image: <built-in function array>
) -> List[Dict[~KT, ~VT]]
```

Run the input image trough YoloV5 Object detection neural network.

Args —= **`input_image`** : Input image as numpy array

Returns: Inference results, list of dictionaries containing bounding boxes,labels and scores

---