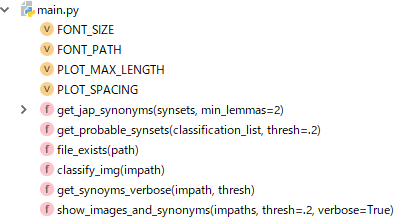
Image Recognition to Japanese Demo

1. Introduction
   1. Welcome to John's image recognition to Japanese code! This demo will explain the basic flow and workings of the program. This program is heavily based on the neural network tutorial found on the tensorflow website: <https://tensorflow.org/tutorials/image_recognition/>  
      The program uses the Inception v3 neural network combined with images from 1000 different categories in the imagenet
2. Installations and Setup
   1. This code is run in Python, constructed using the community version of **Pycharm** I recommend using PyCharm for development and debugging.
3. Flow of the program
   1. Structure of Main:
      1. 
      2. Main is the main method of the program. It takes the imagepaths and outputs a graph of images and Japanese descriptors.
      3. **The program (main) roughly goes as follows:**
         1. Main - \_\_main\_\_ method
            1. show\_images\_and\_synonyms

Initialize graph

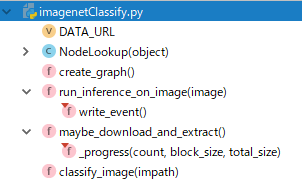
For each image path call:

classify\_img

get\_probable\_synsets

get\_jap\_synonyms

Display images

* + 1. Note: The three looped methods have a verbose version called get\_synonyms\_verbose.
    2. Classify image calls the classify\_image method from the imagenetClassify module.
    3. File\_exists is just a method to check whether a file exists.
    4. Get\_japanese\_synonyms is itself broken down into further functions for ease of debugging and readability.
  1. Structure of classify\_image:
     1. 
     2. imagenetClassify gets called by main.classify\_img.
     3. The main method is classify\_image. It takes an image and guesses the top 5 noun synsets contained in the image, outputting a list of tuples in form (synset\_name, probability, synset\_offset)
     4. **Flow**
        1. classify\_image:
           1. call maybe\_download\_and\_extract to download the neural network if not found locally.

use \_progress to output a progress bar during the download

* + - * 1. call run\_inference\_on\_image

call create\_graph

build the “graph”, or the neural network via data in the classify\_image\_graph\_def.pb protocol buffer file.

send images through the net and output a list of all 1000~ final category nodes and their respective probabilities

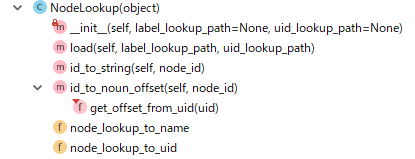
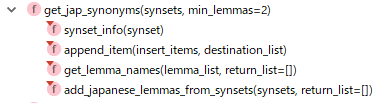
call write event()

writes the event to the tensorboard file TODO: make optional

shrink the output down to only the top n (default 5) categories in the picture

use the NodeLookup class to find the offsets and synset names from the Node\_ID

return the top n categories in form (synset\_name, probability, synset\_offset)

* + - * 1. return the same
  1. Structure of NodeLookup
     1. 
     2. Called by run\_inference\_on\_image. Holds look up tables to look up the network node labels. Nodes each have two labels, one string and one offset. UID is similar to the words’ offsets, but I do not know what it stands for.  
        **Note:** uid – similar to offset, mapping to WordNet  
         id aka nodeid – the node’s specific id number in the neural network
     3. name – string name of the synset
     4. **Structure**
        1. \_\_init\_\_ calls the load method to load the *node\_lookup\_to\_name* dictionary and *node\_lookup\_to\_uid* dictionary each from their respective lookup paths  
           **Note:** The *node\_lookup\_to\_name* dictionary and *node\_lookup\_to\_uid* dictionary are loaded from:  
           'imagenet\_2012\_challenge\_label\_map\_proto.pbtxt' and  
           'imagenet\_synset\_to\_human\_label\_map.txt'  
           respectively. These two files will need to be changed if categories ever want to be changed.
        2. load is the loading method.
        3. id\_to\_string will pass the name of the synset of a specific node to any method that calls NodeLookup().id\_to\_string(node\_id)
        4. id\_to\_noun\_offset will pass the name of the synset of a specific node to any method that calls NodeLookup().id\_to\_string(node\_id)
           1. get\_offset\_from\_uid is a helper function
        5. node\_lookup\_to\_name: dictionary of {node\_id : name}
        6. node\_lookup\_to\_uid: dictionary of {node\_id : uid}
  2. Structure of get\_jap\_synonyms:
     1. 
     2. Called from show\_images\_and\_synonyms
     3. **Flow**:
        1. Passed a list of synsets
        2. Passes all of the Japanese lemmas in the synsets into a variable called *lemmas.*
           1. The lemmas are added by *add\_japanese\_lemmas\_from\_synsets(synsets, [])* from the list of synsets called *synsets* into []
        3. If the number of lemmas found from the synsets is less than *min\_lemmas*, the program gets all the direct hypernyms of *synsets* and makes them the new *synsets* list. It will repeat getting hypernyms and calling the *add\_japanese\_lemmas\_from\_synsets* method adding lemmas from *synsets* into *lemmas* until *min\_lemmas* has been reached.
        4. Lastly the names from all the lemma objects form *synonyms* and *synonyms* is returned.

1. Main Method
   1. Imports
      1. .

*# -\*- coding: utf-8 -\*-***from** pathlib **import** Path  
**from** matplotlib.font\_manager **import** FontProperties  
**from** matplotlib **import** pyplot **as** plt  
**import** math  
**from** modules **import** imagenetClassify  
**from** nltk.corpus **import** wordnet **as** wn

* + 1. This first section of code is at the very top of main.py and contains the import statements. It also contains a header to indicate coding in utf-8 for strings and characters, but I am not sure if this is necessary or not.
       1. Path: This is a library used to check if paths exist or not.
       2. FontProperties: This is a library to edit the font on the chart so that Japanese will be able to come out.
       3. plt: This is the library used to plot pictures and subheadings in a graph-like format.
       4. math: Basic math library
       5. imagenetClassify: A script in the modules folder next to main.
       6. wn: WordNet (Extended Open Multilingual Wordnet) API to work with synsets and translation
  1. Constants
     1. .

*#see description of constants under the show\_images\_and\_synonyms method.*FONT\_SIZE = 10  
FONT\_PATH = **r'C:\WINDOWS\Fonts\yugothm.ttc'**PLOT\_MAX\_LENGTH = 6  
PLOT\_SPACING = 2

* + 1. These are constants that are used in the program. These are moved to the top for ease of editing.
       1. FONT\_SIZE = 10 (font size on the plot)
       2. FONT\_PATH = r'C:\WINDOWS\Fonts\yugothm.ttc' (location of the sinograph-containing font to use)
       3. PLOT\_MAX\_LENGTH = 6 (the max size of rows and columns in the plot)
          1. note: only the first length^2 images from impaths will be shown. For more images, it may be best to write  
             to a file or to run iterations of the program.
       4. PLOT\_SPACING = 2 (interval at which to show images in the plot. I find 1 is too crowded.)
  1. Main method

**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 *# be careful because \U can sometimes mean "Character with 32-bit hex  
 # value xxxxxxxx" - C:\\ is always acceptable  
 # the links need to be in an array* example\_images = [**"C:\\Users\John.McCloskey\PycharmProjects\AnaTensorTest1"  
 "\downloads\spa.jpg"**]  
 show\_images\_and\_synonyms(example\_images, thresh=0)

* + 1. If \_\_name\_\_ == '\_\_main\_\_', then this is this file has been called on its own, not from any other method. In that case, this code is run first. However, it immediately just calls the *show\_images\_and\_synonyms* method.
    2. **Note:** The *show\_images\_and\_synonyms* method needs to be passed a **list** (bracketed collection) of image paths, not just one image path. If these paths are to be passed in bulk, it is recommended that:
       1. the program runs in a loop of batches of images that is less than PLOT\_MAX\_LENGTH^2.
       2. the program reads input data from file and outputs the charts to file.

1. Show\_images\_and\_synonyms
   1. Docstring

**def** show\_images\_and\_synonyms(impaths, thresh=.2, verbose=**True**):

"""  
 Shows images in a square pyplot grid.  
Constants (initialized at the top of the file):  
FONT\_SIZE = 10 (font size on the plot)  
FONT\_PATH = r'C:\WINDOWS\Fonts\yugothm.ttc' (location of the sinograph-containing font to use)  
PLOT\_MAX\_LENGTH = 6 (the max size of rows and columns in the plot)  
---note: only the first length^2 images from impaths will be shown. For more images, it may be best to write  
to a file or to run iterations of the program.

PLOT\_SPACING = 2 (interval at which to show images in the plot. I find 1 is too crowded.)  
**:param** impaths: List<String> list of filepath locations of the images to be read.  
Paths must avoid accidental escape sequences from \ (backslash)  
characters combined with escape characters. Otherwise r can be put  
before strings to indicate a raw string, free of escapes.  
Relative paths must be relative to the location of this program--an image folder is recommended to be put in the  
same folder as "modules".  
**:param** thresh: double. The threshold score for a meaningful categorization in the program. This parameter should be tested  
with different values to see if perhaps a higher or lower value gives better results.  
**:param** verbose: boolean. Indicates whether the program will output text during the program or not. Unless it is a  
nuisance, this should always be True so that users can see scores and categories not listed in the plot.  
"""

* 1. Initializing the Plot

num\_images = len(impaths)  
plot\_length = math.ceil(math.sqrt(num\_images))  
**if** plot\_length > PLOT\_MAX\_LENGTH:  
 plot\_length = PLOT\_MAX\_LENGTH  
**if** plot\_length \* plot\_length < num\_images:  
 num\_images = plot\_length \* plot\_length  
fp = FontProperties(fname=FONT\_PATH, size=FONT\_SIZE)  
plt.figure(figsize=(plot\_length \* PLOT\_SPACING, plot\_length \* PLOT\_SPACING))  
plt.suptitle(**"Thresh = {}"**.format(thresh))

* + 1. Breakdown line-by-line
       1. Find the number of images by length of the aray
       2. Find how many images per row and column in the plot
       3. .
       4. Make sure the plot is not too big
       5. .
       6. Only display as many images as can fit in the plot
       7. Initialize a matplotlib FontProperty object to tell our graph what font to use
          1. the default font does not contain any kanji and cannot be used in Japanese
       8. create a square plot with a side length of

1. Additional Information:
   1. Tensorflow:
      * 1. Introduction:
           1. <https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf>  
              “CS224d: TensorFlow Tutorial” (Powerpoint from Stanford University course CS224d by Bharath Ramsundar). Brief overview of Tensorflow. Very informative. Quick summary:

Tensorflow is just one of many neural network APIs

How to choose an API (Note: at the time, Torch was only available with the Lua language, so Tensorflow was chosen as better because it is written in Python. However, now PyTorch also exists).

a tensor can be represented as a multidimensional array of numbers

**TensorFlow and Numpy are quite similar**

TensorFlow graphs and sessions

Feeds etc.

* + - 1. What is a tensor?
         1. <https://www.tensorflow.org/guide/tensors>  
            “Tensors” by TensorFlow Staff. In-depth information. Covers what a tensor is, what Ranks they can be, their shape, etc.
      2. Tensorboard
      3. Tensorhub
  1. Neural Networks
     + 1. Captioning Neural Nets
          1. Coco
          2. Coco example program
          3. A Hierarchical Approach for Generating Descriptive Image Paragraphs

<https://cs.stanford.edu/people/ranjaykrishna/im2p/index.html>  
Dataset and paper from the Stanford vision lab on captioning pictures

* + - 1. Neural Network Theory
         1. http://cs231n.github.io/  
            github page for the Convolutional Neural Networks for Visual Recognition course at Stanford.
  1. Python and Related Libraries/Software
     + 1. Intro to Python and Libraries
          1. http://cs231n.github.io/python-numpy-tutorial/  
             “Python Numpy Tutorial” from the CS231n Convolutional Neural Networks for Visual Recognition course at Stanford. Python. Numpy. Matplotlib. Subplots. Image display.  
             Very dry. Probably best used as a reference. However, it is a very helpful introduction to Python for experienced programmers.  
             **Note:** This uses scipy.misc to open images, whereas I have been using matplotlib.pyplot. Scipy is supposed to just be a platform that contains matplotlib, so they might indeed be the same method, but I do not know.
       2. CSV library
       3. Anaconda
          1. <https://www.anaconda.com/what-is-anaconda/>
       4. Google Cloud
          1. <http://cs231n.github.io/gce-tutorial/>
          2. <https://cloud.google.com/gcp/>

Google Cloud Homepage. The free trial allows for $300? This looks useful for training networks.

* + - 1. Python C++
      2. url soup
  1. WordNet
     + 1. Japanese WordNet
          1. http://compling.hss.ntu.edu.sg/wnja/  
             “日本語 WordNet” by Linguistics and Multilingual Studies at Nanyang Technological University & 情報通信研究機構. The Python API does not work.
       2. WordNet Structure
          1. link
       3. open multilingual wordnet
  2. ImageNet
     + 1. ImageNet Overview
          1. <http://www.image-net.org/about-overview>  
             © 2016 Stanford Vision Lab, Stanford University, Princeton University

[