Università della Svizzera italiana

Faculty of Informatics

January 13, 2018

Analysis on Transfer Learning

Code Documentation

Sumeet Gyanchandani, Hrishikesh Gupta

1 Code Files

We have six Python 3 files in our project, viz.

1. convert.py
2. resize.py
3. library.py
4. classify.py
5. plot.py
6. main.py

2 Functions

This section would explain the functionality of every function in the project.

2.1 convert.py

- **convert_image**

  Functionality:

  - Receives the path of the file that needs to be converted into a JPEG image, creates the required JPEG image and deletes the old image

  Input Arguments:

  - input_file: The path of the file that needs to be converted into a JPEG image

  Returns:

  - output_file: The path of the converted JPEG image

- **convert**

  Functionality:

  – Iterates over the directory and searches for images that are not JPEG images and call convert_image on them

  Input Arguments:

  – path: The path of the directory/file that needs to be converted into a JPEG images

  Returns:

  – output_path: The path of the converted JPEG image

2.2 resize.py

- **resize_single_file**

  Functionality:

  – Receives the path of the file that needs to be resized, creates the required JPEG image and deletes the old image

  Input Arguments:

  – input_file: The path of the file that needs to be resized

  Returns:

  – output_file: The path of the resized image

- **resize**

  Functionality:

  – Iterates over the directory and call resize_single_file on every image

  Input Arguments:

  – path: The path of the directory/file that needs to be resized

  Returns:

  – output_path: The path of the resized image

2.3 library.py

Functions of this file are originally from the TensorFlow's retraining example using Google's Inception model [1]. We have changed most of them, in order to provide the analysis that we wanted to.

- **create_directory**

  Functionality:

  – Checks whether a given directory is present. If not present, it creates the required directory

  Input Arguments:

  – directory: The path of the directory that needs to be created

- **download_pretrained_model**

  Functionality:

  – If the pretrained model does not exist, it downloads the given model and extracts it

  Input Arguments:

  – MODEL: The URL of the model that needs to be downloaded

- **create_graph**

  Functionality:

  - Creates a graph from the GraphDef file of the downloaded model and returns a Graph object

  Returns:

  - graph: Graph of the pretrained model
  - bottleneck_tensor: The output tensor for the bottleneck values
  - image_data_tensor: The tensor to feed loaded image data into
  - resized_input_tensor: The input node of the recognition graph

- **split_data_into_sets**

  Functionality:

  - Splits the images of all the classes into training, testing, and validation sets and returns the lists of their paths

  Input Arguments:

  - images_count: The number of images per class that are to be included in the sets. (We varied this parameter in order to analyze the accuracy achieved by Transfer Learning)
  - class_count: The number of classes to be considered for classification. (We varied this parameter in order to analyze the accuracy achieved by Transfer Learning)

  Returns:

  - sets: A dictionary containing an entry for each class, with images split into training, testing, and validation sets

- **get_path**

  Functionality:

  - Returns path to a single image

  Input Arguments:

  - directory: Root folder were the classes (subfolders) are stored
  - data_sets: Dictionary of various sets of all the labels
  - class_label: The label for which we need the image
  - index: Offset of the image we want
  - category: The category to pull the image from. Can be either of training, testing, or validation

  Returns:

  - Path to an image that fulfills the input parameters

- **create_bottleneck**

  Functionality:

  - Creates a single bottleneck file. This was very crucial for our analysis, as this is done only once for each file and during our analysis, we had to use each image in several iterations to train the last layer of the graph

  Input Arguments:

  - bottleneck_path: Path to a bottleneck file for a given label
  - data_sets: Dictionary of various sets of all the labels

    - class_label: The label for which we need the image
    - index: Offset of the image we want
    - category: The category to pull the image from. Can be either of training, testing, or validation
    - sess: The currently active TensorFlow Session
    - image_data_tensor: The tensor to feed loaded image data into
    - bottleneck_tensor: The output tensor for the bottleneck values

- **calculate_bottleneck_values**

  Functionality:

    - If we have already saved the bottleneck file, return that. Otherwise, calculate the bottleneck values and save that to a bottleneck file

  Input Arguments:

    - sess: The currently active TensorFlow Session
    - data_sets: Dictionary of various sets for all the labels
    - class_label: The label for which we need the image
    - index: Offset of the image we want
    - category: The category to pull the image from. Can be either of training, testing, or validation
    - image_data_tensor: The tensor to feed loaded image data into
    - bottleneck_tensor: The output tensor for the bottleneck values

  Returns:

    - bottleneck_values: Numpy array of values produced by the bottleneck layer for the image

- **store_bottleneck_values**

  Functionality:

    - In order to provide an analysis on the varied amount of data, we need several iterations of varying number of images per class and number of classes. Hence, we would be reading the same image multiple times. It can speed things up a lot if we calculate the bottleneck layer values once for each image during preprocessing and save them into a file.

  Input Arguments:

    - sess: The currently active TensorFlow Session
    - data_sets: Dictionary of various sets of all the labels
    - image_data_tensor: The tensor to feed loaded image data into
    - bottleneck_tensor: The output tensor for the bottleneck values

- **tensorboard_visualization**

  Functionality:

    - Saves various types of summaries for the given variable for TensorBoard visualization

  Input Arguments:

    - data: The variable for each you would like to see the visualization

- **create_final_layer**

  Functionality:

    - Adds a new softmax and fully-connected layer for training. We need to retrain the top layer to identify our new classes, so this function adds the right operations to the graph, along with some variables to hold the weights, and then sets up all the gradients for the backward pass

Input Arguments:

- class_count: Count of how many categories we are trying to classify
- bottleneck_tensor: The output of the main CNN graph

Returns:

- train_step: The result of a training step
- cross_entropy_mean: The cross entropy result
- bottleneck_input: The bottleneck input tensor
- ground_truth_input: The node into which we will feed the ground truth data
- final_tensor: The final softmax tensor

- **create_evaluation_step**

  Functionality:

  - Inserts the operations that we need to evaluate the accuracy of our results

  Input Arguments:

  - final_tensor: The new final node that produces results
  - ground_truth_tensor: The node into which we will feed the ground truth data

  Returns:

  - evaluation_step
  - prediction

- **load_bottleneck_values**

  Functionality:

  - Returns a random set of previously saved bottleneck values for images from the specified category

  Input Arguments:

  - sess: The currently active TensorFlow Session
  - data_sets: Dictionary of various sets of all the labels
  - batch_size: If positive, a random sample of this size will be retrieved. If negative, all bottlenecks will be retrieved
  - category: The category to pull the image from. Can be either of training, testing, or validation
  - image_data_tensor: The tensor to feed loaded image data into
  - bottleneck_tensor: The output tensor for the bottleneck values
  - images_count: The number of images per class that are to be included in the sets. (We varied this parameter in order to analyze the accuracy achieved by Transfer Learning)
  - class_count: The number of classes to be considered for classification. (We varied this parameter in order to analyze the accuracy achieved by Transfer Learning)

  Returns:

  - bottlenecks: List of bottleneck arrays
  - ground_truths: The ground truths of the bottleneck arrays
  - filenames: The filenames of the bottleneck arrays

- **run_model**

  Functionality:

- Runs the above mentioned functions for retraining, validating and testing the model. It also saves the trained graph into a GraphDef file, which would later be used for prediction

Input Arguments:

- images_count: The number of images per class that are to be included in the sets. (We varied this parameter in order to analyze the accuracy achieved by Transfer Learning)
- class_count: The number of classes to be considered for classification. (We varied this parameter in order to analyze the accuracy achieved by Transfer Learning)
- graph: Graph of the pretrained model
- bottleneck_tensor: The output tensor for the bottleneck values
- image_data_tensor: The tensor to feed loaded image data into
- resized_input_tensor: The input node of the recognition graph

Returns:

- test_accuracy: The final test accuracy of the model

2.4 classify.py

The idea and part of the code in this file belong to Google's codelab: TensorFlow for Poets[2].

- **classify**

  Functionality:

  - Returns a sorted prediction of the classes that the model thinks the image belongs to

  Input Arguments:

  - IMAGE_PATH: The path of the image that we need to classify

2.5 plot.py

The plot function is inspired by Randal Olson's blog: How to make beautiful data visualizations in Python with matplotlib[3].

- **plot**

  Functionality:

  - Plots the result of the analysis

  Input Arguments:

  - RESULT_FILE: The path of the result file that was generated by varying the amount of data

3 Other Parameters

3.1 library.py

- DIRECTORY_MODEL: The path of the model that we need to retrain
- DIRECTORY_TENSORBOARD: The path of the folder where we would like to save the summaries for TensorBoard visualization
- DIRECTORY_BOTTLENECK: The path where we would like to save the bottleneck layer values as files
- DIRECTORY_IMAGES: The path of the root folder that contains the labeled subfolders of images
- BOTTLENECK_TENSOR_NAME: Name of the bottleneck tensor. (This parameter is very specific to the model that we are retraining, in our project it is Google's Inception v3)

- BOTTLENECK_TENSOR_SIZE: Number of entries in the bottleneck tensor. (This parameter is very specific to the model that we are retraining, in our project it is Google's Inception v3)
- IMAGE_DATA_TENSOR_NAME: Name of the image data tensor. (This parameter is very specific to the model that we are retraining, in our project it is Google's Inception v3)
- RESIZED_INPUT_TENSOR_NAME: Name of the resized input tensor. (This parameter is very specific to the model that we are retraining, in our project it is Google's Inception v3)
- PERCENTAGE_TESTING: Percentage of images to use as a test set
- PERCENTAGE_VALIDATION: Percentage of images to use as a validation set
- TRAINING_STEPS: Number of training steps that need to be performed
- VALIDATION_INTERVAL: Interval of how often to evaluate the retraining results
- LEARNING_RATE: The learning rate that we would like to use for the retraining
- TRAINING_BATCH_SIZE: Number of images to train on at a time
- VALIDATION_BATCH_SIZE: Number of images to use in an evaluation batch. This validation set is used much more often than the test set, and is an early indicator of how accurate the model is during training. A value of -1 causes the entire validation set to be used, which leads to more stable results across training iterations, but may be slower on large training sets. Hence, we have always set it equal to the TRAINING_BATCH_SIZE, during our analysis
- TESTING_BATCH_SIZE: Number of images to test on. This test set is only used once, to evaluate the final accuracy of the model after training completes. A negative value causes the entire test set to be used, which leads to more stable results across runs

3.2 classify.py & library.py

- TRAINED_GRAPH: The GraphDef file to which we would like to save the retrained graph
- LABELS: The file to which we would like to save the retrained graph's labels
- FINAL_TENSOR_NAME: The name of the output classification layer in the retrained graph

3.3 resize.py

- size: The dimensions to which we would like to resize the image to. (This is set to the input dimensions of Inception v3, that is, (299, 299))

3.4 main.py

- CONVERT: A boolean flag, which when set to **True**, convert all the image in CONVERT_IMAGE_PATH to JPEG images
- CONVERT_IMAGE_PATH: The path of the image directory that we would like to convert into JPEG images
- RESIZE: A boolean flag, which when set to **True**, resizes all the image in RESIZE_IMAGE_PATH
- RESIZE_IMAGE_PATH: The path of the image directory that we would like to resize
- DOWNLOAD_MODEL: A boolean flag, which when set **True**, downloads the pretrained model specified by MODEL
- MODEL: The URL of the pretrained model that we would like to use
- TRAIN: A boolean flag, which when set to **True**, runs the analysis on the model
- CLASSIFY: A boolean flag, which when set to **True**, runs a classification on the image at CLASSIFY_IMAGE_PATH
- CLASSIFY_IMAGE_PATH: The path of the image that we want to classify
- PLOT: A boolean flag, which when set to **True**, plots the result of the analysis
- RESULT_FILE: The file in which we store the result of the analysis

4 Usage

1. Store all the images of the dataset in the **Images** directory, shown in Figure 1. The subfolders should be labeled as shown in Figure 2.
2. Open **main.py** and change the flags according to the steps that you would like to perform.
3. Run **main.py**.

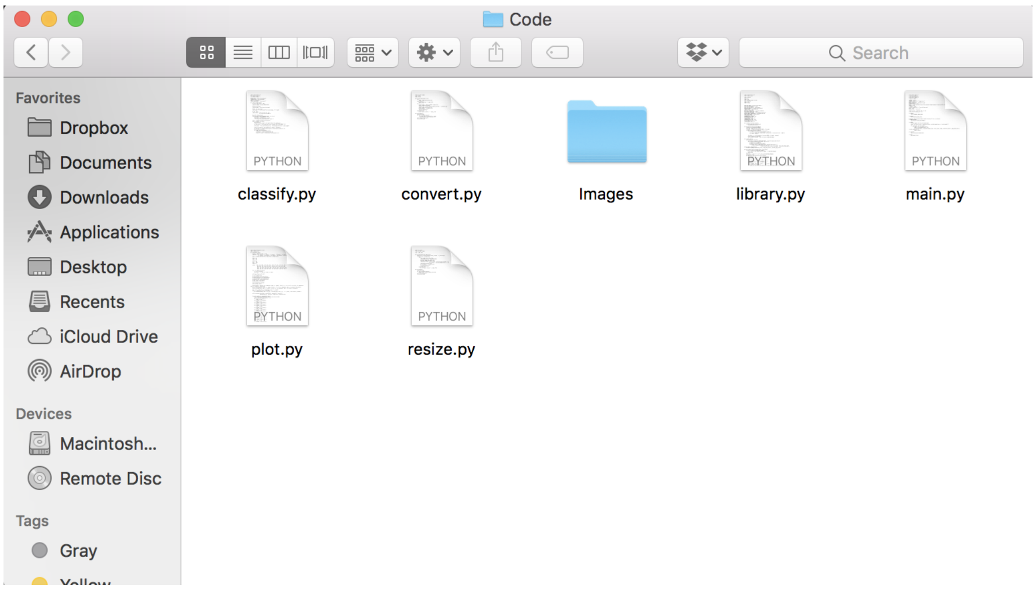

Figure 1. The code directory

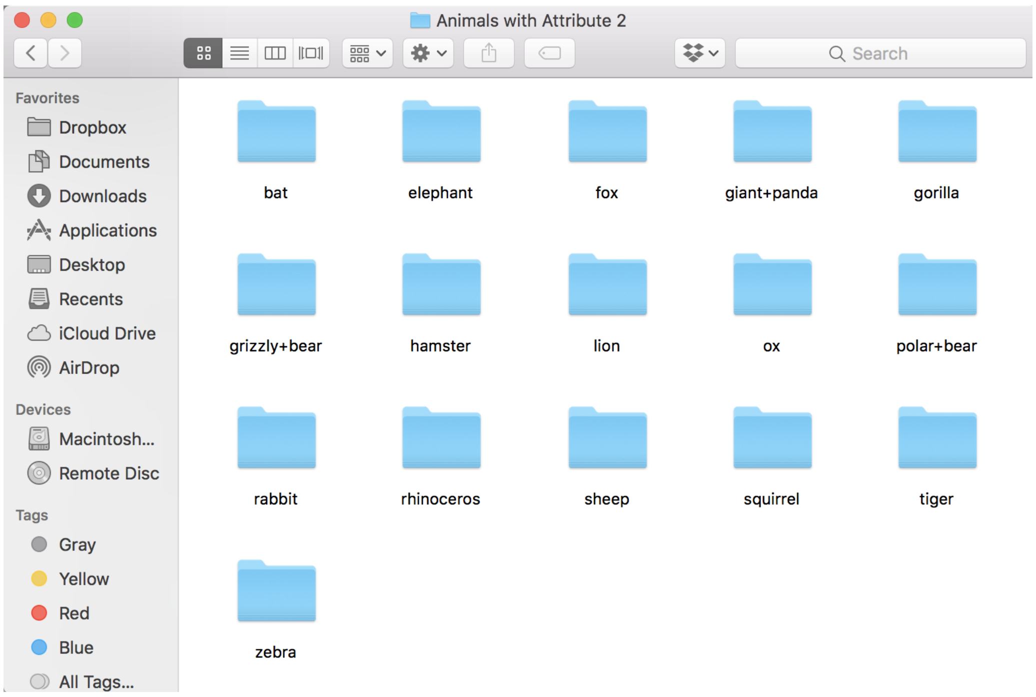

Figure 2. The labeled subfolders

References

[1] TensorFlow, “Tensorflow example on retraining.” https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/image_retraining.

[2] Google, “Tensorflow for poets.” https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0.

[3] R. S. Olson, “How to make beautiful data visualizations in python with matplotlib.” http://www.randalolson.com/2014/06/28/how-to-make-beautiful-data-visualizations-in-python-with-matplotlib/.