ZERO SHOT IMAGE CLASSIFIER

*Meghnad*

Zero Shot Image Classifier Document Version

|  |  |  |
| --- | --- | --- |
| ***Author*** | ***Version*** | ***Date*** |
| *Prakhar Gurha* | *1* | *25.07.2022* |

**Contents**

[1. Introduction 4](#_Toc107399165)

[2. Pre-requisites 4](#_Toc107399166)

* Sample Input and Usage……………………………………………………………………………………………………….5

3. How Does Zero-Shot Learning Work………………………………………………………………………………………….6

[4. Config 7](#_Toc107399167)

[5. Data connectors 7](#_Toc107399168)

[6. Basic Structure of Files 8](#_Toc107399169)

[7. Code Module 8](#_Toc107399170)

[*ZeroShotImgClfConfig* 8](#_Toc107399171)

[ *get\_zsic\_model(self,lang: str=’en’)* 8](#_Toc107399172)

[ *get\_langs\_supported(self)* 8](#_Toc107399173)

[ *get\_zsic\_settings(self)* 8](#_Toc107399174)

[*ZeroShotImageClassification* 9](#_Toc107399175)

[ *available\_models (self)* 9](#_Toc107399176)

[ *available\_languages(self)* 9](#_Toc107399177)

[ *load\_image(self,image:str)* 9](#_Toc107399178)

[ *pred(self,image:str,candidate\_labels:Union[str,List[str]])* 9](#_Toc107399179)

[8. Conclusion 10](#_Toc107399196)

# Introduction

Zero Shot Image classifier, a sub-module of Meghnad under computer vision, is a user-friendly cv algorithm developed where a pre-trained deep learning model is made to generalize on a novel category of samples, i.e., the training and testing set classes are disjoint.

The general idea of zero-shot learning is to transfer the knowledge already contained in the training instances to the task of testing instance classification. Thus, zero-shot learning is a subfield of transfer learning.

This module supports data in multiple formats whether (.jpeg,.jpg,.png, http, https hyperlink) by converting it into a PIL image and the user can input the labels on which the model infers the closest label to the input image as the output.

Some unique features of this module are-

* **Lightweight / faster / conscious labelling needs**
  + Supports CNN based models as vision backbone
* **Multilingual labelling needs**
  + Supports Transformers based models as text backbone for multilingual needs
* **Supported Vision backbones**
  + RN50, RN101, RN50x4, RN50x16, RN50x64, ViT-B/32, ViT-B/16, ViT-L/14
* **Supported Languages**
  + ar, bg, ca, cs, da, de, el, es, et, fa, fi, fr, fr-ca, gl, gu, he, hi, hr, hu, hy, id, it, ja, ka, ko, ku, lt, lv, mk, mn, mr, ms, my, nb, nl, pl, pt, pt, pt-br, ro, ru, sk, sl, sq, sr, sv, th, tr, uk, ur, vi, zh-cn, zh-tw.
* Leverages GPU if available.
* Standing on the shoulder of giants - Open AI CLIP, Sentence-Transformers, Hugging Face Transformers

# Pre-requisites

The input image data on which the labels are predicted in this module should follow some necessary formats. This module uses The Python Imaging Library (PIL) which supports a wide variety of raster file formats. Over 30 different file formats can be identified and read by the library.

The open() function identifies files from their contents, not their names, but save() method looks at the name to determine which format to use, unless the format is given explicitly.

The image formats include .blp,.bmp,.dds,.gif,.jpeg,.jpg .png and many more. For entire list, kindly refer [Image file formats - Pillow (PIL Fork) 9.2.0 documentation](https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html)

In addition, the user can also input the image hyperlink as input to the Zero Shot Classifier.

The prerequisite for this input type is-

* The Image link should start with “http:” or “https:”

**Sample Input and Usage**

1. **Image Input->** <http://images.cocodataset.org/val2017/000000039769.jpg>

**Candidate labels**-> ["cats", "lions", "dogs", "birds"]

**Output->** cats

1. **Image Input->** [https://th.bing.com/th/id/OIP.PODNWnYLw0WNdW- 9aiiGDwHaEK?pid=ImgDet&rs=1](https://th.bing.com/th/id/OIP.PODNWnYLw0WNdW-%20%209aiiGDwHaEK?pid=ImgDet&rs=1)

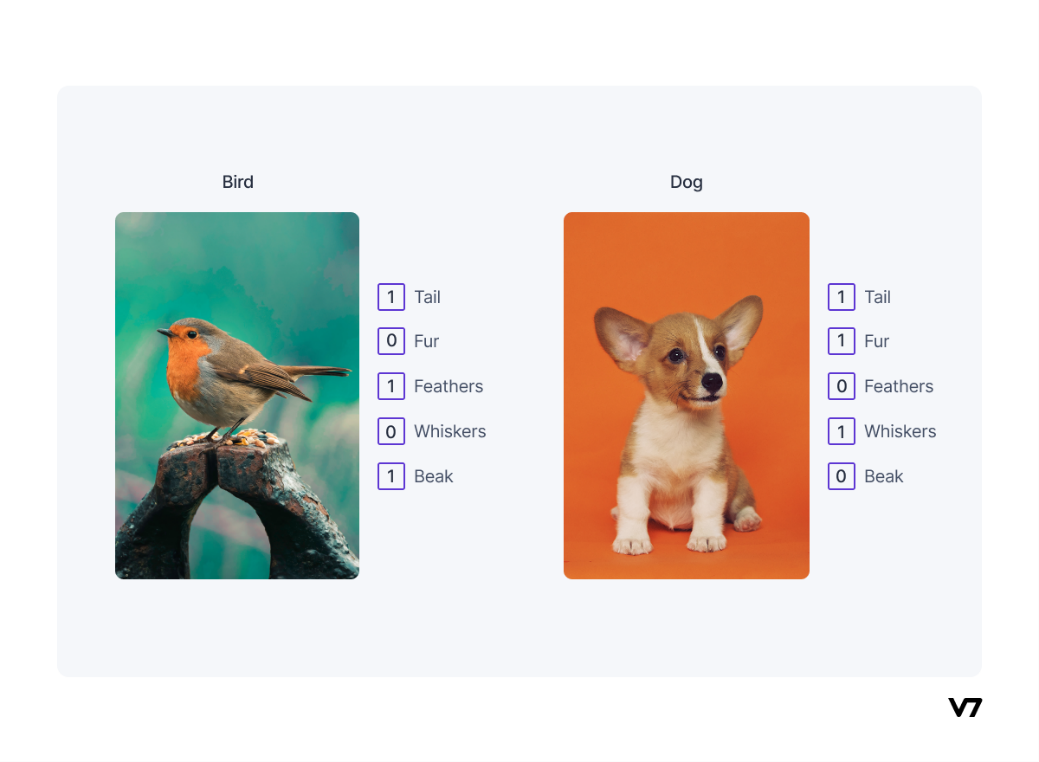
**Candidate Labels**-> ["bear", "polar bear", "fox", "wolf", "brown bear"]

**Output**-> brown bear

# How does Zero-Shot Learning work?

In Zero-Shot Learning, the data consists of the following:

1. **Seen Classes:** These are the data classes that have been used to train the deep learning model.
2. **Unseen Classes:** These are the data classes on which the existing deep model needs to generalize. Data from these classes were not used during training.
3. **Auxiliary Information:** Since no labelled instances belonging to the unseen classes are available, some auxiliary information is necessary to solve the Zero-Shot Learning problem. Such auxiliary information should contain information about all of the unseen classes, which can be descriptions, semantic information, or word embeddings.



Example of semantic embedding using an attribute vector

On the most basic level, Zero-Shot Learning is a two-stage process involving Training and Inference:

1. **Training:**The knowledge about the labelled set of data samples is acquired.
2. **Inference:** The knowledge previously acquired is extended, and the auxiliary information provided is used to the new set of classes.

# Config

Default config settings:

**Language**:’en’

**Model**: 'openai/clip-vit-base-patch-32'

The following languages and models are also available.

* Open AI Clip
* Sentence-transformers
* Supported Vision backbones

RN50, RN101, RN50x4, RN50x16, RN50x64, ViT-B/32, ViT-B/16, ViT-L/14

* **Supported Languages**
  + ar, bg, ca, cs, da, de, el, es, et, fa, fi, fr, fr-ca, gl, gu, he, hi, hr, hu, hy, id, it, ja, ka, ko, ku, lt, lv, mk, mn, mr, ms, my, nb, nl, pl, pt, pt, pt-br, ro, ru, sk, sl, sq, sr, sv, th, tr, uk, ur, vi, zh-cn, zh-tw.

# Data connectors

* **Data path:** Path of train dataset.
* **Data type:** Type of the data file. Currently, all PIL image formats supported
* **Target Cols:** Name of the target column.
* **Feature columns:** List of columns which are to be used as features for training model.
* **Data Org:** Organization of the data.
* **Multi Targets:** If there are multiple target variables, then True else False. Currently not supported.
* **Multi Labels:** If the target variables have multiple labels, then True else False *.*
* **Separator**: Separator of columns in data.
* **Multi Labels Separator:** Separator of labels in target*.*
* **Directory to save model**: Directory where the trained model needs to be saved
* **Model\_mode:** Training mode ('slow', 'medium', 'fast'), Decides the number of default models to be selected while training.

# Basic Structure of Files

The structural format of the files and folders present in the generic classifier module are as below:

# Code Module

## *ZeroShotImgClfConfig*

This class belongs to ‘"Zero\_Shot\_Image\_Classification\cfg\config.py"’. The details of the functions under this class are as follows,

* ***get\_zsic\_model(self, lang:str='en')***

Takes the language code as input (by default ‘en’) and returns the ZSIC model assigned to that language code. Throws an exception if lang code is not among available languages

* ***get\_langs\_supported (self)***

Returns a list of all the language codes supported by ZSIC

* ***get\_zsic\_settings (self)***

Returns a dictionary containing zero shot image classifier settings with multi-value-separation

## 

## *ZeroShotImageClassification*

This class belongs to “Zero\_Shot\_Image\_Classification\src\Zero\_Shot\_Image\_Classification.py"’. The details of the functions under this class are as follows,

### ***available\_models (self)***

This function returns the names of available CLIP models for user to choose from.

### ***available\_languages (self)***

This function returns the codes of available languages for which ZSIC can classify.

codes = ar, bg, ca, cs, da, de, en, el, es, et, fa, fi, fr, fr-ca, gl, gu, he, hi, hr, hu,hy, id, it, ja, ka, ko, ku, lt, lv, mk, mn, mr, ms, my, nb, nl, pl, pt, pt, pt-br, ro, ru, sk, sl, sq, sr, sv, th, tr, uk, ur, vi, zh-cn, zh-tw

### ***load\_image(self, image: str)***

This function loads the input image by handling different kinds of image formats and returns PIL Image for further ZSIC classification on candidate labels

Arguments:

image (`str`):

The image to convert to the PIL Image format.

Returns:

`PIL.Image.Image`: A PIL Image.

Returns a Value Error if Incorrect image format is given as an input. Displays the following message

“*Incorrect format used for image. Should be an url linking to an image, a local path, or a PIL image*.”

### ***pred(self, image: str, candidate\_labels : Union[str, List[str]]])***

This function classifies the image using the candidate labels given and predicts the correct label.

Arguments given:

* image (`str`):

Fully Qualified path of a local image or URL of image

* candidate\_labels (`str` or `List[str]`):

The set of possible class labels to classify each sequence into. Can be a single label, a string of comma-separated labels, or a list of labels

* hypothesis\_template (`str`, \*optional\*, defaults to `"A photo of {}."`, if lang is default / `en`):

The template used to turn each label into a string. This template must include a {} or a similar syntax for the candidate label to be inserted into the template.

* top\_k (`int`, \*optional\*, defaults to 5):

The number of top labels that will be returned by the pipeline. If the provided number is higher than the number of labels available in the model configuration, it will default to 5 as the number of labels.

This function returns:

A `dict` or a list of `dict`: Each result comes as a dictionary with the following keys:

- **\*\*image\*\* (`str`) -- The image for which this is the output.**

**- \*\*labels\*\* (`List[str]`) -- The labels sorted by order of likelihood.**

**- \*\*scores\*\* (`List[float]`) -- The probabilities for each of the labels**.

# Conclusion

Thus, zero shot image classifier module can be used to predict the best label associated with the input image out of the list of candidate labels given by the user. This classifier pipeline enables user to spend the least effort and get the most effective output.

Currently multi-label classifier support is not available in the zero-shot classifier module. This improvement can be enhanced as part of future scope