

# Package ‘AzureVision’

July 21, 2025

**Title** Interface to Azure Computer Vision Services

**Version** 1.0.2

**Description** An interface to 'Azure Computer Vision' <<https://docs.microsoft.com/azure/cognitive-services/Computer-vision/Home>> and 'Azure Custom Vision' <<https://docs.microsoft.com/azure/cognitive-services/custom-vision-service/home>>, building on the low-level functionality provided by the 'AzureCognitive' package. These services allow users to leverage the cloud to carry out visual recognition tasks using advanced image processing models, without needing powerful hardware of their own. Part of the 'AzureR' family of packages.

**URL** <https://github.com/Azure/AzureVision>  
<https://github.com/Azure/AzureR>

**BugReports** <https://github.com/Azure/AzureVision/issues>

**License** MIT + file LICENSE

**VignetteBuilder** knitr

**Depends** R (>= 3.3)

**Imports** AzureRMR, AzureCognitive, httr, utils

**Suggests** knitr, rmarkdown, AzureAuth, testthat

**RoxygenNote** 7.1.1

**NeedsCompilation** no

**Author** Hong Ooi [aut, cre],  
Microsoft [cph]

**Maintainer** Hong Ooi <[hongooi73@gmail.com](mailto:hongooi73@gmail.com)>

**Repository** CRAN

**Date/Publication** 2020-10-17 22:10:03 UTC

## Contents

add_images . . . . .	2
add_image_regions . . . . .	5
add_image_tags . . . . .	7

add_tags . . . . .	8
analyze . . . . .	10
browse_images . . . . .	12
classification_service . . . . .	13
computervision_endpoint . . . . .	14
create_classification_project . . . . .	15
do_training_op . . . . .	17
predict.customvision_model . . . . .	18
publish_model . . . . .	19
show_model . . . . .	21
train_model . . . . .	22

## Index 25

---

add_images	<i>Add, list and remove images for a project</i>
------------	--

---

### Description

Add, list and remove images for a project

### Usage

```
add_images(project, ...)

## S3 method for class 'classification_project'
add_images(project, images, tags = NULL, ...)

## S3 method for class 'object_detection_project'
add_images(project, images, regions = NULL, ...)

list_images(project, include = c("all", "tagged", "untagged"),
  as = c("ids", "dataframe", "list"), iteration = NULL)

remove_images(project, image_ids = list_images(project, "untagged", as =
  "ids"), confirm = TRUE)
```

### Arguments

project	A Custom Vision project.
...	Arguments passed to lower-level functions.
images	For add_images, the images to add (upload) to the project.
tags	Optional tags to add to the images. Only for classification projects.
regions	Optional list of regions in the images that contain objects. Only for object detection projects.
include	For list_images, which images to include in the list: untagged, tagged, or both (the default).

as	For <code>list_images</code> , the return value: a vector of image IDs, a data frame of image metadata, or a list of metadata.
iteration	For <code>list_images</code> , the iteration ID (roughly, which model generation to use). Defaults to the latest iteration.
image_ids	For <code>remove_images</code> , the IDs of the images to remove from the project.
confirm	For <code>remove_images</code> , whether to ask for confirmation first.

## Details

The images to be uploaded can be specified as:

- A vector of local filenames. JPG, PNG and GIF file formats are supported.
- A vector of publicly accessible URLs.
- A raw vector, or a list of raw vectors, holding the binary contents of the image files.

Uploaded images can also have *tags* added (for a classification project) or *regions* (for an object detection project). Classification tags can be specified in the following ways:

- For a regular classification project (one tag per image), as a vector of strings. The tags will be applied to the images in order. If the length of the vector is 1, it will be recycled to the length of `image_ids`.
- For a multilabel classification project (multiple tags per image), as a *list* of vectors of strings. Each vector in the list contains the tags to be assigned to the corresponding image. If the length of the list is 1, it will be recycled to the length of `image_ids`.

If the length of the vector is 1, it will be recycled to the length of `image_ids`.

Object detection projects also have tags, but they are specified as part of the `regions` argument. The regions to add should be specified as a list of data frames, with one data frame per image. Each data frame should have one row per region, and the following columns:

- `left`, `top`, `width`, `height`: the location and dimensions of the region bounding box, normalised to be between 0 and 1.
- `tag`: the name of the tag to associate with the region.

Any other columns in the data frame will be ignored. If the length of the list is 1, it will be recycled to the length of `image_ids`.

Note that once uploaded, images are identified only by their ID; there is no general link back to the source filename or URL. If you don't include tags or regions in the `add_images` call, be sure to save the returned IDs and then call `add_image_tags` or `add_image_regions` as appropriate.

## Value

For `add_images`, the vector of IDs of the uploaded images.

For `list_images`, based on the value of the `as` argument. The default is a vector of image IDs; `as="list"` returns a (nested) list of image metadata with one component per image; and `as="dataframe"` returns the same metadata but reshaped into a data frame.

**See Also**

[add\\_image\\_tags](#) and [add\\_image\\_regions](#) to add tags and regions to images, if not done at upload time

[add\\_tags](#), [list\\_tags](#), [remove\\_tags](#)

[customvision\\_project](#)

**Examples**

```
## Not run:

endp <- customvision_training_endpoint(url="endpoint_url", key="key")

# classification
proj1 <- create_classification_project(endp, "myproject")
list_images(proj1)
imgs <- dir("path/to/images", full.names=TRUE)

# recycling: apply one tag to all images
add_images(proj1, imgs, tags="mytag")
list_images(proj1, include="tagged", as="dataframe")

# different tags per image
add_images(proj1, c("cat.jpg", "dog.jpg", tags=c("cat", "dog"))

# adding online images
host <- "https://mysite.example.com/"
img_urls <- paste0(host, c("img1.jpg", "img2.jpg", "img3.jpg"))
add_images(proj1, img_urls, tags="mytag")

# multiple label classification
proj2 <- create_classification_project(endp, "mymultilabelproject", multiple_tags=TRUE)

add_images(proj2, imgs, tags=list(c("tag1", "tag2")))
add_images(proj2, c("catanddog.jpg", "cat.jpg", "dog.jpg"),
  tags=list(
    c("cat", "dog"),
    "cat",
    "dog"
  )
)

# object detection
proj3 <- create_object_detection_project(endp, "myobjdetproj")

regions <- list(
  data.frame(
    tag=c("cat", "dog"),
    left=c(0.1, 0.5),
    top=c(0.25, 0.28),
    width=c(0.24, 0.21),
    height=c(0.7, 0.6)
```

```
    ),
    data.frame(
      tag="cat", left=0.5, top=0.35, width=0.25, height=0.62
    ),
    data.frame(
      tag="dog", left=0.07, top=0.12, width=0.79, height=0.5
    )
  )
add_images(proj3, c("catanddog.jpg", "cat.jpg", "dog.jpg"), regions=regions)

## End(Not run)
```

---

add\_image\_regions      *Add and remove regions from images*

---

## Description

Add and remove regions from images

## Usage

```
add_image_regions(project, image_ids, regions)

remove_image_regions(project, image_ids, region_ids = NULL)

identify_regions(project, image)
```

## Arguments

project	A Custom Vision object detection project.
image_ids	For add_image_regions and remove_image_regions, the IDs of the images for which to add or remove regions.
regions	For add_image_regions, the regions to add. See 'Details' below.
region_ids	For remove_image_regions, a vector of region IDs. This is an alternative to image ID for specifying the regions to remove; if this is provided, image_ids is not used.
image	For identify_regions, an image for which to identify possible regions in which an object exists. This can be the ID of an image that was previously uploaded to the project; if not, the image is uploaded. Otherwise, see add_images for how to specify an image to upload.

## Details

add\_image\_regions and remove\_image\_regions let you specify the regions in an image that contain an object. You can use identify\_regions to have Custom Vision try to guess the regions for an image.

The regions to add should be specified as a list of data frames, with one data frame per image. Each data frame should have one row per region, and the following columns:

- left, top, width, height: the location and dimensions of the region bounding box, normalised to be between 0 and 1.
- tag: the name of the tag to associate with the region. Any other columns in the data frame will be ignored.

## Value

For add\_image\_regions, a data frame containing the details on the added regions.

For remove\_image\_regions, the value of image\_ids invisibly, if this argument was provided; NULL otherwise.

For identify\_regions, a list with the following components: projectId, the ID of the project; imageId, the ID of the image; and proposals, a data frame containing the coordinates of each identified region along with a confidence score.

## See Also

[add\\_images](#), [add\\_tags](#)

[add\\_image\\_tags](#) for classification projects

## Examples

```
## Not run:
```

```
img_ids <- add_images(myproj, c("catanddog.jpg", "cat.jpg", "dog.jpg"))
```

```
regions <- list(
  data.frame(
    tag=c("cat", "dog"),
    left=c(0.1, 0.5),
    top=c(0.25, 0.28),
    width=c(0.24, 0.21),
    height=c(0.7, 0.6)
  ),
  data.frame(
    tag="cat", left=0.5, top=0.35, width=0.25, height=0.62
  ),
  data.frame(
    tag="dog", left=0.07, top=0.12, width=0.79, height=0.5
  )
)

add_image_regions(myproj, img_ids, regions)
```

```

remove_image_regions(myproj, img_ids[3])
add_image_regions(myproj, img_ids[3],
  list(data.frame(
    tag="dog", left=0.5, top=0.12, width=0.4, height=0.7
  ))
)

## End(Not run)

```

---

add\_image\_tags                      *Tag and untag images uploaded to a project*

---

## Description

Tag and untag images uploaded to a project

## Usage

```

add_image_tags(project, image_ids, tags)

## S3 method for class 'classification_project'
add_image_tags(project, image_ids = list_images(project, "untagged"), tags)

remove_image_tags(project, image_ids = list_images(project, "tagged", as =
  "ids"), tags = list_tags(project, as = "ids"))

```

## Arguments

project	a Custom Vision classification project.
image_ids	The IDs of the images to tag or untag.
tags	For add_image_tags, the tag labels to add to the images. For remove_image_tags, the tags (either text labels or IDs) to remove from images. The default for untagging is to remove all assigned tags.

## Details

add\_image\_tags is for tagging images that were uploaded previously, while remove\_image\_tags untags them. Adding tags does not remove previously assigned ones. Similarly, removing one tag from an image leaves any other tags intact.

Tags can be specified in the following ways:

- For a regular classification project (one tag per image), as a vector of strings. The tags will be applied to the images in order. If the length of the vector is 1, it will be recycled to the length of image\_ids.
- For a multilabel classification project (multiple tags per image), as a *list* of vectors of strings. Each vector in the list contains the tags to be assigned to the corresponding image. If the length of the list is 1, it will be recycled to the length of image\_ids.

If the length of the vector is 1, it will be recycled to the length of image\_ids.

**Value**

The vector of IDs for the images affected, invisibly.

**See Also**

[add\\_images](#), [add\\_tags](#)

[add\\_image\\_regions](#) for object detection projects

**Examples**

```
## Not run:

imgs <- dir("path/to/images", full.names=TRUE)
img_ids <- add_images(myproj, imgs)
add_image_tags(myproj, "mytag")
remove_image_tags(myproj, img_ids[1])
add_image_tags(myproj, img_ids[1], "myothertag")

## End(Not run)
```

---

add\_tags

*Add, retrieve and remove tags for a project*

---

**Description**

Add, retrieve and remove tags for a project

**Usage**

```
add_tags(project, tags)

add_negative_tag(project, negative_name = "_negative_")

list_tags(project, as = c("names", "ids", "dataframe", "list"),
  iteration = NULL)

get_tag(project, name = NULL, id = NULL, iteration = NULL)

remove_tags(project, tags, confirm = TRUE)
```

**Arguments**

project	A Custom Vision project.
tags	For <code>add_tags</code> , a vector of strings to treat as tags.
negative_name	For <code>add_negative_tag</code> , the label to provide a negative tag. See 'Negative tags' below.

as	For <code>list_tags</code> , the format in which to return results: a vector of tag names, a vector of tag IDs, a data frame of metadata, or a list of metadata.
iteration	For <code>list_tags</code> and <code>get_tag</code> , the iteration ID (roughly, which model generation to use). Defaults to the latest iteration.
name, id	For <code>get_tag</code> , the name (text string) for a tag, and its ID. Provide one or the other, but not both.
confirm	For <code>remove_tags</code> , whether to ask for confirmation first.

## Details

*Tags* are the labels attached to images for use in classification projects. An image can have one or multiple tags associated with it; however, the latter only makes sense if the project is setup for multi-label classification.

Tags form part of the metadata for a Custom Vision project, and have to be explicitly defined prior to use. Each tag has a corresponding ID which is used to manage it. In general, you can let AzureVision handle the details of managing tags and tag IDs.

## Value

`add_tags` and `add_negative_tag` return a data frame containing the names and IDs of the tags added.

## Negative tags

A *negative tag* is a special tag that represents the absence of any other tag. For example, if a project is classifying images into cats and dogs, an image that doesn't contain either a cat or dog should be given a negative tag. This can be distinguished from an *untagged* image, where there is no information at all on what it contains.

You can add a negative tag to a project with the `add_negative_tag` method. Once defined, a negative tag is treated like any other tag. A project can only have one negative tag defined.

## See Also

[add\\_image\\_tags](#), [remove\\_image\\_tags](#)

## Examples

```
## Not run:  
  
add_tags(myproj, "newtag")  
add_negative_tag(myproj)  
remove_tags(myproj, "_negative_")  
add_negative_tag(myproj, "nothing")  
  
## End(Not run)
```

---

analyze

---

*Interface to Azure Computer Vision API*


---

## Description

Interface to Azure Computer Vision API

## Usage

```
analyze(endpoint, image, domain = NULL, feature_types = NULL,
        language = "en", ...)

describe(endpoint, image, language = "en", ...)

detect_objects(endpoint, image, ...)

area_of_interest(endpoint, image, ...)

tag(endpoint, image, language = "en", ...)

categorize(endpoint, image, ...)

read_text(endpoint, image, detect_orientation = TRUE, language = "en", ...)

list_computervision_domains(endpoint, ...)

make_thumbnail(endpoint, image, outfile, width = 50, height = 50,
              smart_crop = TRUE, ...)
```

## Arguments

endpoint	A computer vision endpoint.
image	An image to be sent to the endpoint. This can be either a filename, a publicly accessible URL, or a raw vector holding the file contents.
domain	For analyze, an optional domain-specific model to use to analyze the image. Can be "celebrities" or "landmarks".
feature_types	For analyze, an optional character vector of more detailed features to return. This can be one or more of: "categories", "tags", "description", "faces", "im- agetype", "color", "adult", "brands" and "objects". If not supplied, defaults to "categories".
language	A 2-character code indicating the language to use for tags, feature labels and descriptions. The default is en, for English.
...	Arguments passed to lower-level functions, and ultimately to <code>call_cognitive_endpoint</code> .
detect_orientation	For <code>read_text</code> , whether to automatically determine the image's orientation.

outfile	For <code>make_thumbnail</code> , the filename for the generated thumbnail. Alternatively, if this is NULL the thumbnail is returned as a raw vector.
width, height	For <code>make_thumbnail</code> , the dimensions for the returned thumbnail.
smart_crop	For <code>make_thumbnail</code> , whether to automatically determine the best location to crop for the thumbnail. Useful when the aspect ratios of the original image and the thumbnail don't match.

## Details

`analyze` extracts visual features from the image. To obtain more detailed features, specify the `domain` and/or `feature_types` arguments as appropriate.

`describe` attempts to provide a text description of the image.

`detect_objects` detects objects in the image.

`area_of_interest` attempts to find the "interesting" part of an image, meaning the most likely location of the image's subject.

`tag` returns a set of words that are relevant to the content of the image. Not to be confused with the `add_tags` or `add_image_tags` functions that are part of the Custom Vision API.

`categorize` attempts to place the image into a list of predefined categories.

`read_text` performs optical character recognition (OCR) on the image.

`list_domains` returns the predefined domain-specific models that can be queried by `analyze` for deeper analysis. Not to be confused with the domains available for training models with the Custom Vision API.

`make_thumbnail` generates a thumbnail of the image, with the specified dimensions.

## Value

`analyze` returns a list containing the results of the analysis. The components will vary depending on the `domain` and `feature_types` requested.

`describe` returns a list with two components: `tags`, a vector of text labels; and `captions`, a data frame of descriptive sentences.

`detect_objects` returns a dataframe giving the locations and types of the detected objects.

`area_of_interest` returns a length-4 numeric vector, containing the top-left coordinates of the area of interest and its width and height.

`tag` and `categorize` return a data frame of tag and category information, respectively.

`read_text` returns the extracted text as a list with one component per region that contains text. Each component is a vector of character strings.

`list_computervision_domains` returns a character vector of domain names.

`make_thumbnail` returns a raw vector holding the contents of the thumbnail, if the `outfile` argument is NULL. Otherwise, the thumbnail is saved into `outfile`.

## See Also

[computervision\\_endpoint](#), `AzureCognitive::call_cognitive_endpoint`

[Computer Vision documentation](#)

**Examples**

```
## Not run:

vis <- computervision_endpoint(
  url="https://accountname.cognitiveservices.azure.com/",
  key="account_key"
)

list_domains(vis)

# analyze a local file
analyze(vis, "image.jpg")
# picture on the Internet
analyze(vis, "https://example.com/image.jpg")
# as a raw vector
analyze(vis, readBin("image.jpg", "raw", file.size("image.jpg")))

# analyze has optional extras
analyze(vis, "image.jpg", feature_types=c("faces", "objects"))

describe(vis, "image.jpg")
detect_objects(vis, "image.jpg")
area_of_interest(vis, "image.jpg")
tag(vis, "image.jpg") # more reliable than analyze(*, feature_types="tags")
categorize(vis, "image.jpg")
read_text(vis, "scanned_text.jpg")

## End(Not run)
```

---

 browse\_images

 View images uploaded to a Custom Vision project
 

---

**Description**

View images uploaded to a Custom Vision project

**Usage**

```
browse_images(project, img_ids, which = c("resized", "original",
  "thumbnail"), max_images = 20, iteration = NULL)
```

**Arguments**

project	A Custom Vision project.
img_ids	The IDs of the images to view. You can use <a href="#">list_images</a> to get the image IDs for this project.
which	Which image to view: the resized version used for training (the default), the original uploaded image, or the thumbnail.

max_images	The maximum number of images to display.
iteration	The iteration ID (roughly, which model generation to use). Defaults to the latest iteration.

### Details

Images in a Custom Vision project are stored in Azure Storage. This function gets the URLs for the uploaded images and displays them in your browser.

### See Also

[list\\_images](#)

---

classification\_service

*Connect to a Custom Vision predictive service*

---

### Description

Connect to a Custom Vision predictive service

### Usage

```
classification_service(endpoint, project, name)
```

```
object_detection_service(endpoint, project, name)
```

### Arguments

endpoint	A prediction endpoint object, of class <code>customvision_prediction_endpoint</code> .
project	The project underlying this predictive service. Can be either an object of class <code>customvision_project</code> , or a string giving the ID of the project.
name	The published name of the service.

### Details

These functions are handles to a predictive service that was previously published from a trained model. They have predict methods defined for them.

### Value

An object of class `classification_service` or `object_detection_service`, as appropriate. These are subclasses of `customvision_predictive_service`.

**See Also**

[customvision\\_prediction\\_endpoint](#), [customvision\\_project](#)  
[predict.classification\\_service](#), [predict.object\\_detection\\_service](#), [do\\_prediction\\_op](#)  
[train\\_model](#), [publish\\_model](#)

**Examples**

```
## Not run:

endp <- customvision_training_endpoint(url="endpoint_url", key="key")
myproj <- get_project(endp, "myproject")

# getting the ID from the project object -- in practice you would store the ID separately
pred_endp <- customvision_prediction_endpoint(url="endpoint_url", key="pred_key")
classification_service(pred_endp, myproj$project$id, "publishedname")

## End(Not run)
```

---

computervision\_endpoint

*Endpoint objects for computer vision services*

---

**Description**

Endpoint objects for computer vision services

**Usage**

```
computervision_endpoint(url, key = NULL, aad_token = NULL, ...)

customvision_training_endpoint(url, key = NULL, ...)

customvision_prediction_endpoint(url, key = NULL, ...)
```

**Arguments**

<code>url</code>	The URL of the endpoint.
<code>key</code>	A subscription key. Can be single-service or multi-service.
<code>aad_token</code>	For the Computer Vision endpoint, an OAuth token object, of class <a href="#">AzureAuth::AzureToken</a> . You can supply this as an alternative to a subscription key.
<code>...</code>	Other arguments to pass to <a href="#">AzureCognitive::cognitive_endpoint</a> .

## Details

These are functions to create service-specific endpoint objects. Computer Vision supports authentication via either a subscription key or Azure Active Directory (AAD) token; Custom Vision only supports subscription key. Note that there are *two* kinds of Custom Vision endpoint, one for training and the other for prediction.

## Value

An object inheriting from `cognitive_endpoint`. The subclass indicates the type of service/endpoint: Computer Vision, Custom Vision training, or Custom Vision prediction.

## See Also

[cognitive\\_endpoint](#), [call\\_cognitive\\_endpoint](#)

## Examples

```
computervision_endpoint("https://myaccount.cognitiveservices.azure.com", key="key")
customvision_training_endpoint("https://westus.api.cognitive.microsoft.com", key="key")
customvision_prediction_endpoint("https://westus.api.cognitive.microsoft.com", key="key")
```

---

create\_classification\_project

*Create, retrieve, update and delete Azure Custom Vision projects*

---

## Description

Create, retrieve, update and delete Azure Custom Vision projects

## Usage

```
create_classification_project(endpoint, name, domain = "general",
    export_target = c("none", "standard", "vaidk"), multiple_tags = FALSE,
    description = NULL)

create_object_detection_project(endpoint, name, domain = "general",
    export_target = c("none", "standard", "vaidk"), description = NULL)

list_projects(endpoint)

get_project(endpoint, name = NULL, id = NULL)

update_project(endpoint, name = NULL, id = NULL, domain = "general",
    export_target = c("none", "standard", "vaidk"), multiple_tags = FALSE,
```

```
description = NULL)
```

```
delete_project(object, ...)
```

### Arguments

endpoint	A custom vision endpoint.
name, id	The name and ID of the project. At least one of these must be specified for <code>get_project</code> , <code>update_project</code> and <code>delete_project</code> . The name is required for <code>create_project</code> (the ID will be assigned automatically).
domain	What kinds of images the model is meant to apply to. The default "general" means the model is suitable for use in a generic setting. Other, more specialised domains for classification include "food", "landmarks" and "retail"; for object detection the other possible domain is "logo".
export_target	What formats are supported when exporting the model.
multiple_tags	For classification models, Whether multiple categories (tags/labels) for an image are allowed. The default is FALSE, meaning an image represents one and only one category. Ignored for object detection models.
description	An optional text description of the project.
object	For <code>delete_customvision_project</code> , either an endpoint, or a project object.
...	Further arguments passed to lower-level methods.

### Details

A Custom Vision project contains the metadata for a model: its intended purpose (classification vs object detection), the domain, the set of training images, and so on. Once you have created a project, you upload images to it, and train models based on those images. A trained model can then be published as a predictive service, or exported for standalone use.

By default, a Custom Vision project does not support exporting the model; this allows it to be more complex, and thus potentially more accurate. Setting `export_target="standard"` enables exporting to the following formats:

- ONNX 1.2
- CoreML, for iOS 11 devices
- TensorFlow
- TensorFlow Lite, for Android devices
- A Docker image for the Windows, Linux or Raspberry Pi 3 (ARM) platform

Setting `export_target="vaidek"` allows exporting to Vision AI Development Kit format, in addition to the above.

### Value

`delete_project` returns NULL invisibly, on a successful deletion. The others return an object of class `customvision_project`.

**See Also**

[customvision\\_training\\_endpoint](#), [add\\_images](#), [train\\_model](#), [publish\\_model](#), [predict.customvision\\_model](#), [do\\_training\\_op](#)

- [CustomVision.ai](#): An interactive site for building Custom Vision models, provided by Microsoft
- [Training API reference](#)
- [Prediction API reference](#)

**Examples**

```
## Not run:

endp <- customvision_training_endpoint(url="endpoint_url", key="key")

create_classification_project(endp, "myproject")
create_classification_project(endp, "mymultilabelproject", multiple_tags=TRUE)
create_object_detection_project(endp, "myobjdetproj")

create_classification_project(endp, "mystdproject", export_target="standard")

list_projects(endp)

get_project(endp, "myproject")

update_project(endp, "myproject", export_target="vaidk")

## End(Not run)
```

---

<code>do_training_op</code>	<i>Carry out a Custom Vision operation</i>
-----------------------------	--

---

**Description**

Carry out a Custom Vision operation

**Usage**

```
do_training_op(project, ...)

## S3 method for class 'customvision_project'
do_training_op(project, op, ...)

do_prediction_op(service, ...)

## S3 method for class 'customvision_predictive_service'
do_prediction_op(service, op, ...)
```

**Arguments**

project	For do_training_op, a Custom Vision project.
op, ...	Further arguments passed to call_cognitive_endpoint, and ultimately to the REST API.
service	For do_prediction_op, a Custom Vision predictive service.

**Details**

These functions provide low-level access to the Custom Vision REST API. do\_training\_op is for working with the training endpoint, and do\_prediction\_op with the prediction endpoint. You can use them if the other tools in this package don't provide what you need.

**See Also**

[customvision\\_training\\_endpoint](#), [customvision\\_prediction\\_endpoint](#), [customvision\\_project](#), [customvision\\_predictive\\_service](#), [call\\_cognitive\\_endpoint](#)

---

predict.customvision\_model

*Get predictions from a Custom Vision model*

---

**Description**

Get predictions from a Custom Vision model

**Usage**

```
## S3 method for class 'customvision_model'
predict(object, images, type = c("class", "prob", "list"), ...)
```

```
## S3 method for class 'classification_service'
predict(object, images, type = c("class",
  "prob", "list"), save_result = FALSE, ...)
```

```
## S3 method for class 'object_detection_service'
predict(object, images, type = c("class",
  "prob", "list"), save_result = FALSE, ...)
```

**Arguments**

object	A Custom Vision object from which to get predictions. See 'Details' below.
images	The images for which to get predictions.
type	The type of prediction: either class membership (the default), the class probabilities, or a list containing all information returned by the prediction endpoint.
...	Further arguments passed to lower-level functions; not used.
save_result	For the predictive service methods, whether to store the predictions on the server for future use.

## Details

AzureVision defines prediction methods for both Custom Vision model training objects (of class `customvision_model`) and prediction services (`classification_service` and `object_detection_service`). The method for model training objects calls the "quick test" endpoint, and is meant only for testing purposes.

The prediction endpoints accept a single image per request, so supplying multiple images to these functions will call the endpoints multiple times, in sequence. The images can be specified as:

- A vector of local filenames. All common image file formats are supported.
- A vector of publicly accessible URLs.
- A raw vector, or a list of raw vectors, holding the binary contents of the image files.

## See Also

[train\\_model](#), [publish\\_model](#), [classification\\_service](#), [object\\_detection\\_service](#)

## Examples

```
## Not run:

# predicting with the training endpoint
endp <- customvision_training_endpoint(url="endpoint_url", key="key")
myproj <- get_project(endp, "myproject")
mod <- get_model(myproj)

predict(mod, "testimage.jpg")
predict(mod, "https://mysite.example.com/testimage.jpg", type="prob")

imgraw <- readBin("testimage.jpg", "raw", file.size("testimage.jpg"))
predict(mod, imgraw, type="list")

# predicting with the prediction endpoint
# you'll need either the project object or the ID
proj_id <- myproj$project$id
pred_endp <- customvision_prediction_endpoint(url="endpoint_url", key="pred_key")
pred_svc <- classification_service(pred_endp, proj_id, "iteration1")
predict(pred_svc, "testimage.jpg")

## End(Not run)
```

---

publish\_model

*Publish, export and unpublish a Custom Vision model iteration*

---

## Description

Publish, export and unpublish a Custom Vision model iteration

## Usage

```
publish_model(model, name, prediction_resource)

unpublish_model(model, confirm = TRUE)

export_model(model, format, destfile = basename(httr::parse_url(dl_link)$path))

list_model_exports(model)
```

## Arguments

model	A Custom Vision model iteration object.
name	For <code>publish_model</code> , the name to assign to the published model on the prediction endpoint.
prediction_resource	For <code>publish_model</code> , the Custom Vision prediction resource to publish to. This can either be a string containing the Azure resource ID, or an AzureRMR resource object.
confirm	For <code>unpublish_model</code> , whether to ask for confirmation first.
format	For <code>export_model</code> , the format to export to. See below for supported formats.
destfile	For <code>export_model</code> , the destination file for downloading. Set this to NULL to skip downloading.

## Details

Publishing a model makes it available to clients as a predictive service. Exporting a model serialises it to a file of the given format in Azure storage, which can then be downloaded. Each iteration of the model can be published or exported separately.

The `format` argument to `export_model` can be one of the following. Note that exporting a model requires that the project was created with support for it.

- "onnx": ONNX 1.2
- "coreml": CoreML, for iOS 11 devices
- "tensorflow": TensorFlow
- "tensorflow lite": TensorFlow Lite for Android devices
- "linux docker", "windows docker", "arm docker": A Docker image for the given platform (Raspberry Pi 3 in the case of ARM)
- "vaidk": Vision AI Development Kit

## Value

`export_model` returns the URL of the exported file, invisibly if it was downloaded.

`list_model_exports` returns a data frame detailing the formats the current model has been exported to, along with their download URLs.

**See Also**

[train\\_model](#), [get\\_model](#), [customvision\\_predictive\\_service](#), [predict.classification\\_service](#), [predict.object\\_detection\\_service](#)

**Examples**

```
## Not run:

endp <- customvision_training_endpoint(url="endpoint_url", key="key")
myproj <- get_project(endp, "myproject")
mod <- get_model(myproj)

export_model(mod, "tensorflow", download=FALSE)
export_model(mod, "onnx", destfile="onnx.zip")

rg <- AzureRMR::get_azure_login("yourtenant")$
  get_subscription("sub_id")$
  get_resource_group("rgname")

pred_res <- rg$get_cognitive_service("mycustvis_prediction")
publish_model(mod, "mypublishedmod", pred_res)

unpublish_model(mod)

## End(Not run)
```

---

show\_model

*Display model iteration details*


---

**Description**

Display model iteration details

**Usage**

```
show_model(model)

show_training_performance(model, threshold = 0.5, overlap = NULL)

## S3 method for class 'customvision_model'
summary(object, ...)
```

**Arguments**

`model`, `object`    A Custom Vision model iteration object.  
`threshold`        For a classification model, the probability threshold to assign an image to a class.

overlap	For an object detection model, the overlap threshold for distinguishing between overlapping objects.
...	Arguments passed to lower-level functions.

### Details

show\_model displays the metadata for a model iteration: the name (assigned by default), model training status, publishing details, and so on. show\_training\_performance displays summary statistics for the model's performance on the training data. The summary method for Custom Vision model objects simply calls show\_training\_performance.

### Value

For show\_model, a list containing the metadata for the model iteration. For show\_training\_performance and summary.customvision\_model, a list of performance diagnostics.

### See Also

[train\\_model](#)

### Examples

```
## Not run:

endp <- customvision_training_endpoint(url="endpoint_url", key="key")
myproj <- get_project(endp, "myproject")
mod <- get_model(myproj)

show_model(mod)

show_training_performance(mod)
summary(mod)

## End(Not run)
```

---

train\_model

*Create, retrieve, rename and delete a model iteration*

---

### Description

Create, retrieve, rename and delete a model iteration

**Usage**

```

train_model(project, training_method = c("quick", "advanced"),
  max_time = 1, force = FALSE, email = NULL, wait = (training_method ==
    "quick"))

list_models(project, as = c("ids", "list"))

get_model(project, iteration = NULL)

rename_model(model, name, ...)

delete_model(object, ...)

## S3 method for class 'customvision_project'
delete_model(object, iteration = NULL, confirm = TRUE, ...)

## S3 method for class 'customvision_model'
delete_model(object, confirm = TRUE, ...)

```

**Arguments**

project	A Custom Vision project.
training_method	The training method to use. The default "quick" is faster but may be less accurate. The "advanced" method is slower but produces better results.
max_time	For advanced training, the maximum training time in hours.
force	For advanced training, whether to refit the model even if the data has not changed since the last iteration.
email	For advanced training, an email address to notify when the training is complete.
wait	whether to wait until training is complete (or the maximum training time has elapsed) before returning.
as	For list_models, the format in which to return results: as a named vector of model iteration IDs, or a list of model objects.
iteration	For get_model and delete_model.customvision_project, either the iteration name or ID.
model	A Custom Vision model.
name	For rename_model, the new name for the model.
...	Arguments passed to lower-level functions.
object	For the delete_model method, a Custom Vision project or model, as appropriate.
confirm	For the delete_model methods, whether to ask for confirmation first.

## Details

Training a Custom Vision model results in a *model iteration*. Each iteration is based on the current set of images uploaded to the endpoint. Successive model iterations trained on different image sets do not overwrite previous ones.

You must have at least 5 images per tag for a classification project, and 15 images per tag for an object detection project, before you can train a model.

By default, AzureVision will use the latest model iteration for actions such as prediction, showing performance statistics, and so on. You can list the model iterations with `list_models`, and retrieve a specific iteration by passing the iteration ID to `get_model`.

## Value

For `train_model`, `get_model` and `rename_model`, an object of class `customvision_model` which is a handle to the iteration.

For `list_models`, based on the `as` argument: `as="ids"` returns a named vector of model iteration IDs, while `as="list"` returns a list of model objects.

## See Also

[show\\_model](#), [show\\_training\\_performance](#), [publish\\_model](#)

## Examples

```
## Not run:

endp <- customvision_training_endpoint(url="endpoint_url", key="key")
myproj <- get_project(endp, "myproject")

train_model(myproj)
train_model(myproj, method="advanced", force=TRUE, email="me@example.com")

list_models(myproj)

mod <- get_model(myproj)
rename(mod, "mymodel")
mod <- get_model(myproj, "mymodel")

delete_model(mod)

## End(Not run)
```

# Index

add\_image\_regions, [3](#), [4](#), [5](#), [8](#)  
add\_image\_tags, [3](#), [4](#), [6](#), [7](#), [9](#), [11](#)  
add\_images, [2](#), [6](#), [8](#), [17](#)  
add\_negative\_tag (add\_tags), [8](#)  
add\_tags, [4](#), [6](#), [8](#), [8](#), [11](#)  
analyze, [10](#)  
area\_of\_interest (analyze), [10](#)  
AzureAuth::AzureToken, [14](#)  
AzureCognitive::call\_cognitive\_endpoint,  
[11](#)  
AzureCognitive::cognitive\_endpoint, [14](#)  
  
browse\_images, [12](#)  
  
call\_cognitive\_endpoint, [15](#), [18](#)  
categorize (analyze), [10](#)  
classification\_service, [13](#), [19](#)  
cognitive\_endpoint, [15](#)  
computervision (analyze), [10](#)  
computervision\_endpoint, [11](#), [14](#)  
create\_classification\_project, [15](#)  
create\_object\_detection\_project  
(create\_classification\_project),  
[15](#)  
customvision\_image\_tags  
(add\_image\_tags), [7](#)  
customvision\_images (add\_images), [2](#)  
customvision\_prediction\_endpoint, [14](#),  
[18](#)  
customvision\_prediction\_endpoint  
(computervision\_endpoint), [14](#)  
customvision\_predictive\_service, [18](#), [21](#)  
customvision\_predictive\_service  
(classification\_service), [13](#)  
customvision\_project, [4](#), [14](#), [18](#)  
customvision\_project  
(create\_classification\_project),  
[15](#)  
customvision\_regions  
(add\_image\_regions), [5](#)  
  
customvision\_tags (add\_tags), [8](#)  
customvision\_training\_endpoint, [17](#), [18](#)  
customvision\_training\_endpoint  
(computervision\_endpoint), [14](#)  
  
delete\_model (train\_model), [22](#)  
delete\_project  
(create\_classification\_project),  
[15](#)  
describe (analyze), [10](#)  
detect\_objects (analyze), [10](#)  
do\_prediction\_op, [14](#)  
do\_prediction\_op (do\_training\_op), [17](#)  
do\_training\_op, [17](#), [17](#)  
  
export\_model (publish\_model), [19](#)  
  
get\_model, [21](#)  
get\_model (train\_model), [22](#)  
get\_project  
(create\_classification\_project),  
[15](#)  
get\_tag (add\_tags), [8](#)  
  
identify\_regions (add\_image\_regions), [5](#)  
  
list\_computervision\_domains (analyze),  
[10](#)  
list\_images, [12](#), [13](#)  
list\_images (add\_images), [2](#)  
list\_model\_exports (publish\_model), [19](#)  
list\_models (train\_model), [22](#)  
list\_projects  
(create\_classification\_project),  
[15](#)  
list\_tags, [4](#)  
list\_tags (add\_tags), [8](#)  
  
make\_thumbnail (analyze), [10](#)  
  
object\_detection\_service, [19](#)

object\_detection\_service  
    (classification\_service), 13

predict (predict.customvision\_model), 18  
predict.classification\_service, 14, 21  
predict.customvision\_model, 17, 18  
predict.object\_detection\_service, 14,  
    21

publish\_model, 14, 17, 19, 19, 24

read\_text (analyze), 10

remove\_image\_regions  
    (add\_image\_regions), 5

remove\_image\_tags, 9

remove\_image\_tags (add\_image\_tags), 7

remove\_images (add\_images), 2

remove\_tags, 4

remove\_tags (add\_tags), 8

rename\_model (train\_model), 22

show\_model, 21, 24

show\_training\_performance, 24

show\_training\_performance (show\_model),  
    21

summary.customvision\_model  
    (show\_model), 21

tag (analyze), 10

train\_model, 14, 17, 19, 21, 22, 22

unpublish\_model (publish\_model), 19

update\_project  
    (create\_classification\_project),  
    15