Cloud AutoML Vision Object Detection

Method: projects.locations.models.batchPredict

Perform a batch prediction. Unlike the online `models.predict` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models/predict#google.cloud.automl.v1beta1.PredictionService.Predict) , batch prediction result won't be immediately available in the response. Instead, a long running operation object is returned. User can poll the operation result via `operations.get` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.operations/get#google.longrunning.Operations.GetOperation) method. Once the operation is done, `BatchPredictResult` is returned in the `response` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.operations#Operation.FIELDS.response) field. Available for following ML problems: * Image Classification * Image Object Detection * Video Classification * Video Object Tracking * Text Extraction * Tables

HTTP request

```
POST https://automl.googleapis.com/v1beta1/{name}:batchPredict
```

Path parameters

Parameters	
`name`	`string` Name of the model requested to serve the batch prediction. Authorization requires the following Google IAM (https://cloud.google.com/iam) permission on the specified resource **name**: • `automl.models.predict`

Request body

The request body contains data with the following structure:

JSON representation

```
{
  "inputConfig": {
    object (BatchPredictInputConfig (https://cloud.google.com/vision/automl/object-detection/doc
  },
  "outputConfig": {
    object (BatchPredictOutputConfig (https://cloud.google.com/vision/automl/object-detection/do
  },
  "params": {
    string: string,
    ...
  }
}
```

Fields	
`inputConfig`	`object (BatchPredictInputConfig` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models/batchPredict#BatchPredictInputConfig) `)` Required. The input configuration for batch prediction.
`outputConfig`	`object (BatchPredictOutputConfig` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models/batchPredict#BatchPredictOutputConfig) `)` Required. The Configuration specifying where output predictions should be written.
`params`	`map (key: string, value: string)` Additional domain-specific parameters for the predictions, any string must be up to 25000 characters long. • For Text Classification:

Fields

`score_threshold` - (float) A value from 0.0 to 1.0. When the model makes predictions for a text snippet, it will only produce results that have at least this confidence score. The default is 0.5.

- For Image Classification:

`score_threshold` - (float) A value from 0.0 to 1.0. When the model makes predictions for an image, it will only produce results that have at least this confidence score. The default is 0.5.

- For Image Object Detection:

`score_threshold` - (float) When Model detects objects on the image, it will only produce bounding boxes which have at least this confidence score. Value in 0 to 1 range, default is 0.5. `max_bounding_box_count` - (int64) No more than this number of bounding boxes will be produced per image. Default is 100, the requested value may be limited by server.

- For Video Classification : `score_threshold` - (float) A value from 0.0 to 1.0. When the model makes predictions for a video, it will only produce results that have at least this confidence score. The default is 0.5. `segment_classification` - (boolean) Set to true to request segment-level classification. AutoML Video Intelligence returns labels and their confidence scores for the entire segment of the video that user specified in the request configuration. The default is "true". `shot_classification` - (boolean) Set to true to request shot-level classification. AutoML Video Intelligence determines the boundaries for each camera shot in the entire segment of the video that user specified in the request configuration. AutoML Video Intelligence then returns labels and their confidence scores for each detected shot, along with the start and end time of the shot. WARNING: Model evaluation is not done for this classification type, the quality of it depends on training data, but there are no metrics provided to describe that quality. The default is "false". `1s_interval_classification` - (boolean) Set to true to request classification for a video at one-second intervals. AutoML Video Intelligence returns labels and their confidence scores for each second of the entire segment of the video that user specified in the request configuration. WARNING: Model evaluation is not done for this classification type, the quality of it depends on training data, but there are no metrics provided to describe that quality. The default is "false".
- For Video Object Tracking: `score_threshold` - (float) When Model detects objects on video frames, it will only produce bounding boxes

Fields	
	which have at least this confidence score. Value in 0 to 1 range, default is 0.5. `max_bounding_box_count` - (int64) No more than this number of bounding boxes will be returned per frame. Default is 100, the requested value may be limited by server. `min_bounding_box_size` - (float) Only bounding boxes with shortest edge at least that long as a relative value of video frame size will be returned. Value in 0 to 1 range. Default is 0. An object containing a list of `"key": value` pairs. Example: `{ "name": "wrench", "mass": "1.3kg", "count": "3" }`.

Response body

If successful, the response body contains an instance of `Operation` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.operations#Operation)
.

Authorization Scopes

Requires the following OAuth scope:

- `https://www.googleapis.com/auth/cloud-platform`

For more information, see the Authentication Overview (https://cloud.google.com/docs/authentication/).

BatchPredictInputConfig

Input configuration for models.batchPredict Action.

The format of input depends on the ML problem of the model used for prediction. As input source the `gcsSource`

(https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs#InputConfig.FIELDS.gcs_source)
is expected, unless specified otherwise.

The formats are represented in EBNF with commas being literal and with non-terminal symbols defined near the end of this comment. The formats are:

- For Image Classification: CSV file(s) with each line having just a single column: GCS_FILE_PATH which leads to image of up to 30MB in size. Supported extensions: .JPEG, .GIF, .PNG. This path is treated as the ID in the Batch predict output. Three sample rows: gs://folder/image1.jpeg (gs://folder/image1.jpeg) gs://folder/image2.gif (gs://folder/image2.gif) gs://folder/image3.png (gs://folder/image3.png)
- For Image Object Detection: CSV file(s) with each line having just a single column: GCS_FILE_PATH which leads to image of up to 30MB in size. Supported extensions: .JPEG, .GIF, .PNG. This path is treated as the ID in the Batch predict output. Three sample rows: gs://folder/image1.jpeg (gs://folder/image1.jpeg) gs://folder/image2.gif (gs://folder/image2.gif) gs://folder/image3.png (gs://folder/image3.png)
- For Video Classification: CSV file(s) with each line in format: GCS_FILE_PATH,TIME_SEGMENT_START,TIME_SEGMENT_END GCS_FILE_PATH leads to video of up to 50GB in size and up to 3h duration. Supported extensions: .MOV, .MPEG4, .MP4, .AVI. TIME_SEGMENT_START and TIME_SEGMENT_END must be within the length of the video, and end has to be after the start. Three sample rows: gs://folder/video1.mp4,10,40 (gs://folder/video1.mp4,10,40) gs://folder/video1.mp4,20,60 (gs://folder/video1.mp4,20,60) gs://folder/vid2.mov,0,inf (gs://folder/vid2.mov,0,inf)
- For Video Object Tracking: CSV file(s) with each line in format: GCS_FILE_PATH,TIME_SEGMENT_START,TIME_SEGMENT_END GCS_FILE_PATH leads to video of up to 50GB in size and up to 3h duration. Supported extensions: .MOV, .MPEG4, .MP4, .AVI. TIME_SEGMENT_START and TIME_SEGMENT_END must be within the length of the video, and end has to be after the start. Three sample rows: gs://folder/video1.mp4,10,240 (gs://folder/video1.mp4,10,240) gs://folder/video1.mp4,300,360 (gs://folder/video1.mp4,300,360) gs://folder/vid2.mov,0,inf (gs://folder/vid2.mov,0,inf)
- For Text Classification: CSV file(s) with each line having just a single column: GCS_FILE_PATH | TEXT_SNIPPET Any given text file can have size upto 128kB. Any given text snippet content must have 60,000 characters or less. Three sample rows:

gs://folder/text1.txt (gs://folder/text1.txt) "Some text content to predict"
gs://folder/text3.pdf (gs://folder/text3.pdf) Supported file extensions: .txt, .pdf

- For Text Sentiment: CSV file(s) with each line having just a single column: GCS_FILE_PATH | TEXT_SNIPPET Any given text file can have size upto 128kB. Any given text snippet content must have 500 characters or less. Three sample rows: gs://folder/text1.txt (gs://folder/text1.txt) "Some text content to predict" gs://folder/text3.pdf (gs://folder/text3.pdf) Supported file extensions: .txt, .pdf
- For Text Extraction .JSONL (i.e. JSON Lines) file(s) which either provide text in-line or as documents (for a single models.batchPredict call only one of the these formats may be used). The in-line .JSONL file(s) contain per line a proto that wraps a temporary user-assigned TextSnippet ID (string up to 2000 characters long) called "id", a TextSnippet proto (in json representation) and zero or more TextFeature protos. Any given text snippet content must have 30,000 characters or less, and also be UTF-8 NFC encoded (ASCII already is). The IDs provided should be unique. The document .JSONL file(s) contain, per line, a proto that wraps a Document proto with inputConfig set. Only PDF documents are supported now, and each document must be up to 2MB large. Any given .JSONL file must be 100MB or smaller, and no more than 20 files may be given. Sample in-line JSON Lines file (presented here with artificial line breaks, but the only actual line break is denoted by \n): { "id": "my_first_id", "textSnippet": { "content": "dog car cat"}, "text_features": [{ "textSegment": {"startOffset": 4, "endOffset": 6}, "structural_type": PARAGRAPH, "boundingPoly": { "normalizedVertices": [{"x": 0.1, "y": 0.1}, {"x": 0.1, "y": 0.3}, {"x": 0.3, "y": 0.3}, {"x": 0.3, "y": 0.1},] }, }], }\n { "id": "2", "textSnippet": { "content": "An elaborate content", "mimeType": "text/plain" } } Sample document JSON Lines file (presented here with artificial line breaks, but the only actual line break is denoted by \n).: { "document": { "inputConfig": { "gcsSource": { "inputUris": ["gs://folder/document1.pdf"] } } } }\n { "document": { "inputConfig": { "gcsSource": { "inputUris": ["gs://folder/document2.pdf"] } } } }
- For Tables: Either `gcsSource` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs#InputConfig.FIELDS.gcs_source)
  or

`bigquerySource`. GCS case: CSV file(s), each by itself 10GB or smaller and total size must be 100GB or smaller, where first file must have a header containing column names. If the first row of a subsequent file is the same as the header, then it is also treated as a header. All other rows contain values for the corresponding columns. The column names must contain the model's `inputFeatureColumnSpecs'` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models#TablesModelMetadata.FIELDS.input_feature_column_specs)

`displayName-s` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs.columnSpecs#ColumnSpec.FIELDS.display_name)
(order doesn't matter). The columns corresponding to the model's input feature column specs must contain values compatible with the column spec's data types. Prediction on all the rows, i.e. the CSV lines, will be attempted. For FORECASTING

`predictionType`: all columns having

`TIME_SERIES_AVAILABLE_PAST_ONLY` type will be ignored. First three sample rows of a CSV file:
"First Name","Last Name","Dob","Addresses"

"John","Doe","1968-01-22","
[{"status":"current","address":"123_First_Avenue","city":"Seattle","state":"WA","zip":"11111","number
{"status":"previous","address":"456_Main_Street","city":"Portland","state":"OR","zip":"22222","numbe

"Jane","Doe","1980-10-16","
[{"status":"current","address":"789_Any_Avenue","city":"Albany","state":"NY","zip":"33333","numberO
{"status":"previous","address":"321_Main_Street","city":"Hoboken","state":"NJ","zip":"44444","numbe
BigQuery case: An URI of a BigQuery table. The user data size of the BigQuery table must be 100GB or smaller. The column names must contain the model's

`inputFeatureColumnSpecs'` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models#TablesModelMetadata.FIELDS.input_feature_column_specs)

`displayName-s` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs.columnSpecs#ColumnSpec.FIELDS.display_name)
(order doesn't matter). The columns corresponding to the model's input feature column specs must contain values compatible with the column spec's data types. Prediction on all the rows of the table will be attempted. For FORECASTING

`predictionType`: all columns having

`TIME_SERIES_AVAILABLE_PAST_ONLY` type will be ignored.

Definitions: GCS_FILE_PATH = A path to file on GCS, e.g. "gs://folder/video.avi". TEXT_SNIPPET = A content of a text snippet, UTF-8 encoded, enclosed within double quotes ("") TIME_SEGMENT_START = TIME_OFFSET Expresses a beginning, inclusive, of a time segment within an example that has a time dimension (e.g. video). TIME_SEGMENT_END = TIME_OFFSET Expresses an end, exclusive, of a time segment within an example that has a time dimension (e.g. video). TIME_OFFSET = A number of seconds as measured from the start of an example (e.g. video). Fractions are allowed, up to a microsecond precision. "inf" is allowed and it means the end of the example.

Errors: If any of the provided CSV files can't be parsed or if more than certain percent of CSV rows cannot be processed then the operation fails and prediction does not happen. Regardless of overall success or failure the per-row failures, up to a certain count cap, will be listed in Operation.metadata.partial_failures.

JSON representation

```
{
  "gcsSource": {
    object (GcsSource (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v
  }
}
```

Fields	
`gcsSource`	`object (GcsSource` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs#GcsSource) `)` The Google Cloud Storage location for the input content.

BatchPredictOutputConfig

Output configuration for models.batchPredict Action.

As destination the

`gcsDestination` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models/batchPredict#BatchPredictOutputConfig.FIELDS.gcs_destination)
must be set unless specified otherwise for a domain. If gcsDestination is set then in the given directory a new directory is created. Its name will be "prediction–", where timestamp is in YYYY-MM-DDThh:mm:ss.sssZ ISO-8601 format. The contents of it depends on the ML problem the predictions are made for.

- For Image Classification: In the created directory files `image_classification_1.jsonl`, `image_classification_2.jsonl`,…,`image_classification_N.jsonl` will be created, where N may be 1, and depends on the total number of the successfully predicted images and annotations. A single image will be listed only once with all its annotations, and its annotations will never be split across files. Each .JSONL file will contain, per line, a JSON representation of a proto that wraps image's "ID" : "" followed by a list of zero or more AnnotationPayload protos (called annotations), which have classification detail populated. If prediction for any image failed (partially or completely), then an additional `errors_1.jsonl`, `errors_2.jsonl`,…, `errors_N.jsonl` files will be created (N depends on total number of failed predictions). These files will have a JSON representation of a proto that wraps the same "ID" : "" but here followed by exactly one

[`google.rpc.Status`](https: //github.com/googleapis/googleapis/blob/master/google/rpc/status.proto) containing only `code` and `message`fields.

- For Image Object Detection: In the created directory files `image_object_detection_1.jsonl`, `image_object_detection_2.jsonl`,…,`image_object_detection_N.jsonl` will be created, where N may be 1, and depends on the total number of the successfully predicted images and annotations. Each .JSONL file will contain, per line, a JSON representation of a proto that wraps image's "ID" : "" followed by a list of zero or more AnnotationPayload protos (called annotations), which have imageObjectDetection detail populated. A single image will be listed only once with all its annotations, and its annotations will never be split across files. If prediction for any image failed (partially or completely), then additional `errors_1.jsonl`, `errors_2.jsonl`,…, `errors_N.jsonl` files will be created (N depends on total number of failed predictions). These files will have a JSON representation of a proto that wraps the same "ID" : "" but here followed by exactly one [`google.rpc.Status`](https://github.com/googleapis/googleapis/blob/master/google/rpc/status.proto) containing only `code` and `message`fields. * For Video Classification: In the created directory a videoClassification.csv file, and a .JSON file per each video classification requested in the input (i.e. each line in given CSV(s)), will be created.

```
The format of videoClassification.csv is:
```

GCS_FILE_PATH,TIME_SEGMENT_START,TIME_SEGMENT_END,JSON_FILE_NAME,STATUS where: GCS_FILE_PATH,TIME_SEGMENT_START,TIME_SEGMENT_END = matches 1 to 1 the prediction input lines (i.e. videoClassification.csv has precisely the same number of lines as the prediction input had.) JSON_FILE_NAME = Name of .JSON file in the output directory, which contains prediction responses for the video time segment. STATUS = "OK" if prediction completed successfully, or an error code with message otherwise. If STATUS is not "OK" then the .JSON file for that line may not exist or be empty.

```
Each .JSON file, assuming STATUS is "OK", will contain a list of
AnnotationPayload protos in JSON format, which are the predictions
for the video time segment the file is assigned to in the
videoClassification.csv. All AnnotationPayload protos will have
videoClassification field set, and will be sorted by
videoClassification.type field (note that the returned types are
governed by `classifaction_types` parameter in
[PredictService.BatchPredictRequest.params][]).
```

- For Video Object Tracking: In the created directory a videoObjectTracking.csv file will be created, and multiple files video_object_trackinng_1.json, video_object_trackinng_2.json,..., video_object_trackinng_N.json, where N is the number of requests in the input (i.e. the number of lines in given CSV(s)).

```
The format of videoObjectTracking.csv is:
```

GCS_FILE_PATH,TIME_SEGMENT_START,TIME_SEGMENT_END,JSON_FILE_NAME,STATUS where: GCS_FILE_PATH,TIME_SEGMENT_START,TIME_SEGMENT_END = matches 1 to 1 the prediction input lines (i.e. videoObjectTracking.csv has precisely the same number of lines as the prediction input had.) JSON_FILE_NAME = Name of .JSON file in the output directory, which contains prediction responses for the video time segment. STATUS = "OK" if prediction completed successfully, or an error code with message otherwise. If STATUS is not "OK" then the .JSON file for that line may not exist or be empty.

```
Each .JSON file, assuming STATUS is "OK", will contain a list of
AnnotationPayload protos in JSON format, which are the predictions
for each frame of the video time segment the file is assigned to in
videoObjectTracking.csv. All AnnotationPayload protos will have
videoObjectTracking field set.
```

- For Text Classification: In the created directory files `text_classification_1.jsonl`, `text_classification_2.jsonl`,...,`text_classification_N.jsonl` will be created, where N may be 1, and depends on the total number of inputs and annotations found.

```
Each .JSONL file will contain, per line, a JSON representation of a
proto that wraps input text snippet or input text file and a list of
zero or more AnnotationPayload protos (called annotations), which
have classification detail populated. A single text snippet or file
will be listed only once with all its annotations, and its
annotations will never be split across files.

If prediction for any text snippet or file failed (partially or
completely), then additional `errors_1.jsonl`, `errors_2.jsonl`,...,
`errors_N.jsonl` files will be created (N depends on total number of
failed predictions). These files will have a JSON representation of a
proto that wraps input text snippet or input text file followed by
exactly one
```

[`google.rpc.Status`](https:
//github.com/googleapis/googleapis/blob/master/google/rpc/status.proto) containing only `code` and `message`.

- For Text Sentiment: In the created directory files `text_sentiment_1.jsonl`, `text_sentiment_2.jsonl`,...,`text_sentiment_N.jsonl` will be created, where N may be 1, and depends on the total number of inputs and annotations found.

```
Each .JSONL file will contain, per line, a JSON representation of a
proto that wraps input text snippet or input text file and a list of
zero or more AnnotationPayload protos (called annotations), which
have textSentiment detail populated. A single text snippet or file
will be listed only once with all its annotations, and its
annotations will never be split across files.

If prediction for any text snippet or file failed (partially or
completely), then additional `errors_1.jsonl`, `errors_2.jsonl`,...,
`errors_N.jsonl` files will be created (N depends on total number of
failed predictions). These files will have a JSON representation of a
proto that wraps input text snippet or input text file followed by
exactly one
```

[`google.rpc.Status`](https://github.com/googleapis/googleapis/blob/master/google/rpc/status.proto) containing only `code` and `message`.

- For Text Extraction: In the created directory files `text_extraction_1.jsonl`, `text_extraction_2.jsonl`,...,`text_extraction_N.jsonl` will be created, where N may be 1, and depends on the total number of inputs and annotations found. The contents of these .JSONL file(s) depend on whether the input used inline text, or documents. If input was inline, then each .JSONL file will contain, per line, a JSON representation of a proto that wraps given in request text snippet's "id" (if specified), followed by input text snippet, and a list of zero or more AnnotationPayload protos (called annotations), which have textExtraction detail populated. A single text snippet will be listed only once with all its annotations, and its annotations will never be split across files. If input used documents, then each .JSONL file will contain, per line, a JSON representation of a proto that wraps given in request document proto, followed by its OCR-ed representation in the form of a text snippet, finally followed by a list of zero or more AnnotationPayload protos (called annotations), which have textExtraction detail populated and refer, via their indices, to the OCR-ed text snippet. A single document (and its text snippet) will be listed only once with all its annotations, and its annotations will never be split across files. If prediction for any text snippet failed (partially or completely), then additional `errors_1.jsonl`, `errors_2.jsonl`,..., `errors_N.jsonl` files will be created (N depends on total number of failed predictions). These files will have a JSON representation of a proto that wraps either the "id" : "" (in case of inline) or the document proto (in case of document) but here followed by exactly one

[`google.rpc.Status`](https://github.com/googleapis/googleapis/blob/master/google/rpc/status.proto) containing only `code` and `message`.

- For Tables: Output depends on whether

`gcsDestination`
 (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models/batchPredict#BatchPredictOutputConfig.FIELDS.gcs_destination)
or

`bigqueryDestination` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models/batchPredict#BatchPredictOutputConfig.FIELDS.bigquery_destination) is set (either is allowed). GCS case: In the created directory files `tables_1.csv`, `tables_2.csv`,..., `tables_N.csv` will be created, where N may be 1, and depends on the total number of the successfully predicted rows. For all CLASSIFICATION

`predictionType-s`: Each .csv file will contain a header, listing all columns'

`displayName-s` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs.columnSpecs#ColumnSpec.FIELDS.display_name) given on input followed by M target column names in the format of

"<`target_column_specs` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models#TablesModelMetadata.FIELDS.target_column_spec)

`displayName` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs.columnSpecs#ColumnSpec.FIELDS.display_name) >__score" where M is the number of distinct target values, i.e. number of distinct values in the target column of the table used to train the model. Subsequent lines will contain the respective values of successfully predicted rows, with the last, i.e. the target, columns having the corresponding prediction `scores` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models/predict#TablesAnnotation.FIELDS.score) . For REGRESSION and FORECASTING

`predictionType-s`: Each .csv file will contain a header, listing all columns' [displayName-s] [google.cloud.automl.v1beta1.display_name] given on input followed by the predicted target column with name in the format of

"predicted_<`target_column_specs` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models#TablesModelMetadata.FIELDS.target_column_spec)

`displayName`
(https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs.columnSpecs#ColumnSpec.FIELDS.display_name)
>" Subsequent lines will contain the respective values of successfully predicted rows, with the last, i.e. the target, column having the predicted target value. If prediction for any rows failed, then an additional `errors_1.csv`, `errors_2.csv`,..., `errors_N.csv` will be created (N depends on total number of failed rows). These files will have analogous format as `tables_*.csv`, but always with a single target column having

[`google.rpc.Status`](https://github.com/googleapis/googleapis/blob/master/google/rpc/status.proto) represented as a JSON string, and containing only `code` and `message`. BigQuery case:

`bigqueryDestination`
(https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.datasets/exportData#OutputConfig.FIELDS.bigquery_destination)
pointing to a BigQuery project must be set. In the given project a new dataset will be created with name `prediction_<model-display-name>_<timestamp-of-prediction-call>` where will be made BigQuery-dataset-name compatible (e.g. most special characters will become underscores), and timestamp will be in YYYY_MM_DDThh_mm_ss_sssZ "based on ISO-8601" format. In the dataset two tables will be created, `predictions`, and `errors`. The `predictions` table's column names will be the input columns'

`displayName-s`
(https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs.columnSpecs#ColumnSpec.FIELDS.display_name)
followed by the target column with name in the format of

"predicted_<`target_column_specs`
(https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models#TablesModelMetadata.FIELDS.target_column_spec)

`displayName`
(https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs.columnSpecs#ColumnSpec.FIELDS.display_name)

>" The input feature columns will contain the respective values of successfully predicted rows, with the target column having an ARRAY of

`AnnotationPayloads` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models/predict#AnnotationPayload)
, represented as STRUCT-s, containing `TablesAnnotation` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models/predict#TablesAnnotation)
. The `errors` table contains rows for which the prediction has failed, it has analogous input columns while the target column name is in the format of

"errors_<`target_column_specs` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.models#TablesModelMetadata.FIELDS.target_column_spec)

`displayName` (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/projects.locations.datasets.tableSpecs.columnSpecs#ColumnSpec.FIELDS.display_name)
>", and as a value has

[`google.rpc.Status`](https:
//github.com/googleapis/googleapis/blob/master/google/rpc/status.proto) represented as a STRUCT, and containing only `code` and `message`.

JSON representation

```
{

  // Union field destination can be only one of the following:
  "gcsDestination": {
    object (GcsDestination (https://cloud.google.com/vision/automl/object-detection/docs/reference
  },
  "bigqueryDestination": {
    object (BigQueryDestination (https://cloud.google.com/vision/automl/object-detection/docs/refe
  }
  // End of list of possible types for union field destination.
}
```

Fields

Fields	
Union field `destination`. Required. The destination of the output. `destination` can be only one of the following:	
`gcsDestination`	`object (`[`GcsDestination`](https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/GcsDestination) (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/GcsDestination) `)` The Google Cloud Storage location of the directory where the output is to be written to.
`bigqueryDestination`	`object (`[`BigQueryDestination`](https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/BigQueryDestination) (https://cloud.google.com/vision/automl/object-detection/docs/reference/rest/v1beta1/BigQueryDestination) `)` The BigQuery location where the output is to be written to.