Batch image annotation offline

The Vision API can run offline (asynchronous) detection services and annotation of a large batch of image files using any Vision API feature type (https://cloud.google.com/vision/docs/reference/rest/v1/Feature#type). For example, you can specify one or multiple Vision API features (such as `TEXT_DETECTION`, `LABEL_DETECTION`, and `LANDMARK_DETECTION`) for a single batch of images.

Output from an offline batch request is written to a JSON file created in the specified Cloud Storage bucket.

Note: All Vision API feature types (https://cloud.google.com/vision/docs/reference/rest/v1/Feature#type) also offer **online synchronous batch image annotation**. Online synchronous requests offer an immediate response to an annotation request. For more information, refer to the how-to (https://cloud.google.com/vision/docs/how-to) pages.

- **online (synchronous)** requests - An online annotation request (`images:annotate` or `files:annotate`) immediately returns inline annotations to the user. Online annotation requests limit the amount of files you can annotate in a single request; generally you can only specify a single file or small amount of files (<=5) to be annotated.
- **offline (asynchronous)** requests - An offline annotation request (`images:asyncBatchAnnotate` or `files:asyncBatchAnnotate`) starts a long-running operation (LRO) and does not immediately return a response to the caller. When the LRO completes, annotations are stored as files in a Cloud Storage bucket you specify. Offline annotation requests allow you to specify larger batches of files (<=2000) for annotation at a single time than you are able to with online requests.

Limitations

The Vision API accepts up to 2,000 image files. A larger batch of image files will return an error.

Currently supported feature types

Feature type	
CROP_HINTS (https://cloud.google.com/vision/docs/detecting-crop-hints)	Determine suggested vertices for a crop region on an image.
DOCUMENT_TEXT_DETECTION (https://cloud.google.com/vision/docs/pdf)	Perform OCR on dense text images, such as documents (PDF/TIFF), and images with handwriting. **TEXT_DETECTION** can be used for sparse text images. Takes precedence when both **DOCUMENT_TEXT_DETECTION** and **TEXT_DETECTION** are present.
FACE_DETECTION (https://cloud.google.com/vision/docs/detecting-faces)	Detect faces within the image.
IMAGE_PROPERTIES (https://cloud.google.com/vision/docs/detecting-properties)	Compute a set of image properties, such as the image's dominant colors.
LABEL_DETECTION (https://cloud.google.com/vision/docs/labels)	Add labels based on image content.
LANDMARK_DETECTION (https://cloud.google.com/vision/docs/detecting-landmarks)	Detect geographic landmarks within the image.
LOGO_DETECTION (https://cloud.google.com/vision/docs/detecting-logos)	Detect company logos within the image.
OBJECT_LOCALIZATION (https://cloud.google.com/vision/docs/object-localizer)	Detect and extract multiple objects in an image.
SAFE_SEARCH_DETECTION (https://cloud.google.com/vision/docs/detecting-safe-search)	Run Safe Search to detect potentially unsafe or undesirable content.

Feature type	
TEXT_DETECTION (https://cloud.google.com/vision/docs/ocr)	Perform Optical Character Recognition (OCR) on text within the image. Text detection is optimized for areas of sparse text within a larger image. If the image is a document (PDF/TIFF), has dense text, or contains handwriting, use **DOCUMENT_TEXT_DETECTION** instead.
WEB_DETECTION (https://cloud.google.com/vision/docs/detecting-web)	Detect topical entities such as news, events, or celebrities within the image, and find similar images on the web using the power of Google Image Search.

Sample code

Use the following code samples to run offline annotation services on a batch of image files in Cloud Storage.

Note: In the following code samples each requests element (`requests_element`/`requestsElement`) corresponds to a single image. To annotate more images, create a request element for each image and add it to the array of requests (`requests`).

JAVA NODE.JS PYTHON RUBY

Before trying this sample, follow the Java setup instructions in the Vision API Quickstart Using Client Libraries (https://cloud.google.com/vision/docs/quickstart-client-libraries). For more information, see the Vision API Java API reference documentation (https://googleapis.dev/java/google-cloud-vision/latest).

Note: For Java **Spring framework** users, Spring Cloud GCP (https://cloud.google.com/vision/docs/adding-spring) offers a way to automatically configure authentication settings and client objects to use Vision API.

LOUD-CLIENT/SRC/MAIN/JAVA/COM/EXAMPLE/VISION/VISIONASYNCBATCHANNOTATEIMAGES.JAVA)

FEEDBACK (#)

```
/*
 * Please include the following imports to run this sample.
 *
```

```
 * import com.google.cloud.vision.v1.AnnotateImageRequest;
 * import com.google.cloud.vision.v1.AsyncBatchAnnotateImagesRequest;
 * import com.google.cloud.vision.v1.AsyncBatchAnnotateImagesResponse;
 * import com.google.cloud.vision.v1.Feature;
 * import com.google.cloud.vision.v1.GcsDestination;
 * import com.google.cloud.vision.v1.Image;
 * import com.google.cloud.vision.v1.ImageAnnotatorClient;
 * import com.google.cloud.vision.v1.ImageSource;
 * import com.google.cloud.vision.v1.OutputConfig;
 * import java.util.Arrays;
 * import java.util.List;
 */

/** Perform async batch image annotation */
public static void sampleAsyncBatchAnnotateImages(String inputImageUri, String out
  try (ImageAnnotatorClient imageAnnotatorClient = ImageAnnotatorClient.create())
    // inputImageUri = "gs://cloud-samples-data/vision/label/wakeupcat.jpg";
    // outputUri = "gs://your-bucket/prefix/";
    ImageSource source = ImageSource.newBuilder().setImageUri(inputImageUri).build
    Image image = Image.newBuilder().setSource(source).build();
    Feature.Type type = Feature.Type.LABEL_DETECTION;
    Feature featuresElement = Feature.newBuilder().setType(type).build();
    Feature.Type type2 = Feature.Type.IMAGE_PROPERTIES;
    Feature featuresElement2 = Feature.newBuilder().setType(type2).build();
    List<Feature> features = Arrays.asList(featuresElement, featuresElement2);
    AnnotateImageRequest requestsElement =
        AnnotateImageRequest.newBuilder().setImage(image).addAllFeatures(features)
    List<AnnotateImageRequest> requests = Arrays.asList(requestsElement);
    GcsDestination gcsDestination = GcsDestination.newBuilder().setUri(outputUri).

    // The max number of responses to output in each JSON file
    int batchSize = 2;
    OutputConfig outputConfig =
        OutputConfig.newBuilder()
            .setGcsDestination(gcsDestination)
            .setBatchSize(batchSize)
            .build();
    AsyncBatchAnnotateImagesRequest request =
        AsyncBatchAnnotateImagesRequest.newBuilder()
            .addAllRequests(requests)
            .setOutputConfig(outputConfig)
            .build();
    AsyncBatchAnnotateImagesResponse response =
        imageAnnotatorClient.asyncBatchAnnotateImagesAsync(request).get();
    // The output is written to GCS with the provided output_uri as prefix
```

```
    String gcsOutputUri = response.getOutputConfig().getGcsDestination().getUri();
    System.out.printf("Output written to GCS with prefix: %s\n", gcsOutputUri);
  } catch (Exception exception) {
    System.err.println("Failed to create the client due to: " + exception);
  }
}
```

Response

A successful request returns response JSON files in the Cloud Storage bucket you indicated in the code sample. The number of responses per JSON file is dictated by `batch_size` in the code sample.

The returned response is similar to regular Cloud Vision API feature responses, depending on which features you request for an image.

The following responses show `LABEL_DETECTION` and `TEXT_DETECTION` annotations for `image1.png`, `IMAGE_PROPERTIES` annotations for `image2.jpg`, and `OBJECT_LOCALIZATION` annotations for `image3.jpg`.

The response also contain a `context` field showing the file's URI.

offline_batch_output/output-1-to-2.json

```
{
  "responses": [
    {
      "labelAnnotations": [
        {
          "mid": "/m/07s6nbt",
          "description": "Text",
          "score": 0.93413997,
          "topicality": 0.93413997
        },
        {
          "mid": "/m/0dwx7",
          "description": "Logo",
          "score": 0.8733531,
          "topicality": 0.8733531
        },
        ...
```

```
      {
        "mid": "/m/03bxgrp",
        "description": "Company",
        "score": 0.5682425,
        "topicality": 0.5682425
      }
    ],
    "textAnnotations": [
      {
        "locale": "en",
        "description": "Google\n",
        "boundingPoly": {
          "vertices": [
            {
              "x": 72,
              "y": 40
            },
            {
              "x": 613,
              "y": 40
            },
            {
              "x": 613,
              "y": 233
            },
            {
              "x": 72,
              "y": 233
            }
          ]
        }
      },
      ...
              ],
              "blockType": "TEXT"
            }
          ]
        }
      ],
      "text": "Google\n"
    },
    "context": {
      "uri": "gs://cloud-samples-data/vision/document_understanding/image1.png"
    }
  },
```

```
{
  "imagePropertiesAnnotation": {
    "dominantColors": {
      "colors": [
        {
          "color": {
            "red": 229,
            "green": 230,
            "blue": 238
          },
          "score": 0.2744754,
          "pixelFraction": 0.075339235
        },
        ...
        {
          "color": {
            "red": 86,
            "green": 87,
            "blue": 95
          },
          "score": 0.025770646,
          "pixelFraction": 0.13109145
        }
      ]
    }
  },
  "cropHintsAnnotation": {
    "cropHints": [
      {
        "boundingPoly": {
          "vertices": [
            {},
            {
              "x": 1599
            },
            {
              "x": 1599,
              "y": 1199
            },
            {
              "y": 1199
            }
          ]
        },
        "confidence": 0.79999995,
```

```
            "importanceFraction": 1
          }
        ]
      },
      "context": {
        "uri": "gs://cloud-samples-data/vision/document_understanding/image2.jpg"
      }
    }
  ]
}
```

offline_batch_output/output-3-to-3.json

```
{
  "responses": [
    {
      "context": {
        "uri": "gs://cloud-samples-data/vision/document_understanding/image3.jpg"
      },
      "localizedObjectAnnotations": [
        {
          "mid": "/m/0bt9lr",
          "name": "Dog",
          "score": 0.9669734,
          "boundingPoly": {
            "normalizedVertices": [
              {
                "x": 0.6035543,
                "y": 0.1357359
              },
              {
                "x": 0.98546547,
                "y": 0.1357359
              },
              {
                "x": 0.98546547,
                "y": 0.98426414
              },
              {
                "x": 0.6035543,
                "y": 0.98426414
              }
            ]
          }
```

```
        },
        ...
        {
          "mid": "/m/0jbk",
          "name": "Animal",
          "score": 0.58003056,
          "boundingPoly": {
            "normalizedVertices": [
              {
                "x": 0.014534635,
                "y": 0.1357359
              },
              {
                "x": 0.37197515,
                "y": 0.1357359
              },
              {
                "x": 0.37197515,
                "y": 0.98426414
              },
              {
                "x": 0.014534635,
                "y": 0.98426414
              }
            ]
          }
        }
      ]
    }
  ]
}
```


Last updated January 6, 2020.