



Siddhartha Lama

# Vertex AI Image Classification Pipeline Using Cloud Functions and AutoML

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology ( Smart IoT system )

Internship Project Report

28 February 2024

## Abstract

Author: Siddhartha Lama  
Title: Cloud Function AutoML Image Classification Pipeline  
Number of Pages: xx pages + x appendices  
Date: 28 Feb 2024

Degree: Bachelor of Engineering  
Degree Programme: Information Technology  
Professional Major: Smart IoT systems  
Supervisors: Erkki Räsänen, Lecturer  
Fuwad Kalhori, Project Manager

---

### Abstract:

Construction site waste management presents a critical challenge globally, necessitating innovative solutions to minimize environmental impact and promote sustainability. This project addresses this challenge by developing an AI model integrated into a mobile app, aimed at assisting construction site workers in identifying and selling partially used or unused construction materials. Leveraging Google Cloud's infrastructure, the model is trained using a pipeline employing incremental learning, also known as transfer learning, to continuously improve classification accuracy. The mobile app allows users to capture images of materials and predicts the material in the image. The images are also uploaded to a Google Cloud bucket for future use as a dataset. The model's accuracy improves over time as more images are collected, facilitating efficient waste management, and contributing to the reduction of construction site waste.

Keywords: Construction waste, AI mobile app, Google Cloud, Incremental learning, Sustainability.

# Contents

List of Abbreviations

## Table of Contents

1	Background	1
2	Introduction	3
2.1	Objective	3
3	System Design	5
4	Implementation	6
4.1	Step-by-step Description of the Pipeline Development	6
4.1.1	Pushing Images to GCP Bucket	6
4.1.2	Cloud Function – 1	8
4.1.3	Cloud Function - 2	11
4.1.4	Deploying the Model	12
4.1.5	Making Inferences	14
4.1.6	Moving 90 % of Images and Deleting tag.csv	14
5	Discussion and Cost Analysis.	15
5.1	Model Training Cost Analysis	15
5.2	Model Deployment Cost Analysis	17
5.3	Cloud Function Costs	20
5.3.1	Coud Function Invocations Costs	20
5.3.2	Cloud Function Deployment Cost:	20
6	Results	22
7	Conclusions	22
	References	23

## **List of Abbreviations**

GCP: Google Cloud Platform

GUI: Graphical User Interface

GCS: Google Cloud Storage

AI: Artificial Intelligence

## 1 Background

Construction site waste is a significant global issue that has profound impacts on the environment. The construction industry generates vast amounts of waste, including materials like concrete, wood, metals, plastics, and more. Improper disposal of these materials leads to pollution of land, air, and water, contributing to environmental degradation and posing risks to ecosystems and human health.

The impacts of construction waste on the environment are multifaceted. Firstly, the extraction and manufacturing processes required to produce construction materials deplete natural resources and generate greenhouse gas emissions. Secondly, the disposal of waste in landfills contributes to the release of harmful chemicals and greenhouse gases into the atmosphere. Additionally, construction waste can contaminate soil and water sources, leading to long-term environmental damage and ecosystem disruption.

To address this issue, innovative solutions are needed. One such solution involves the integration of robust and user-friendly apps equipped with customized AI models to detect and categorize construction site materials accurately. By utilizing machine learning algorithms, these apps can identify salvageable materials from construction sites, allowing them to be repurposed or sold as second-hand products.

This approach offers several benefits. Firstly, it reduces the amount of waste sent to landfills, mitigating environmental pollution and conserving valuable resources. Secondly, it promotes the circular economy by facilitating the reuse and recycling of construction materials, thereby reducing the demand for new raw materials and lowering carbon emissions associated with production processes.

Moreover, the integration of AI technology streamlines the identification and sorting process, making it more efficient and cost-effective for construction companies and waste management organizations. By providing a user-friendly platform accessible to stakeholders across the construction industry, including contractors, builders, and recyclers, the app encourages collaboration and facilitates the exchange of materials, ultimately contributing to the reduction of construction waste on a global scale.

In conclusion, the effective management of construction site waste is crucial for mitigating its adverse impacts on the environment. By harnessing the power of technology, such as AI-integrated apps, to identify and repurpose construction materials, we can promote sustainability, minimize waste generation, and create a more resilient and resource-efficient construction industry.

## 2 Introduction

Construction site waste management is a significant global issue, demanding innovative approaches to lessen environmental impacts and enhance sustainability. This project tackles this challenge by developing an advanced AI model, designed to be integrated into a mobile application specifically for construction site personnel. The aim is to aid workers in identifying and facilitating the sale of partially used or unused construction materials, thereby promoting recycling, and reducing waste.

There are different types of image classification models offered by GCP in AutoML, such as Single label image classification model, Multi label image classification model, object detection model, and image segmentation model. In our case, looking at the material images taken by the users in the field, we could either train a single label image classification model or an object detection model. But it is just a matter of choice; one can easily switch between models using the GUI or by changing the name of the trained model in a cloud function. However, the data preparation for the object detection model requires a much more tedious task of manually labelling the products in each image with bounding boxes. Depending on the use case, one can switch from one model to another before the training starts.

### 2.1 Objective

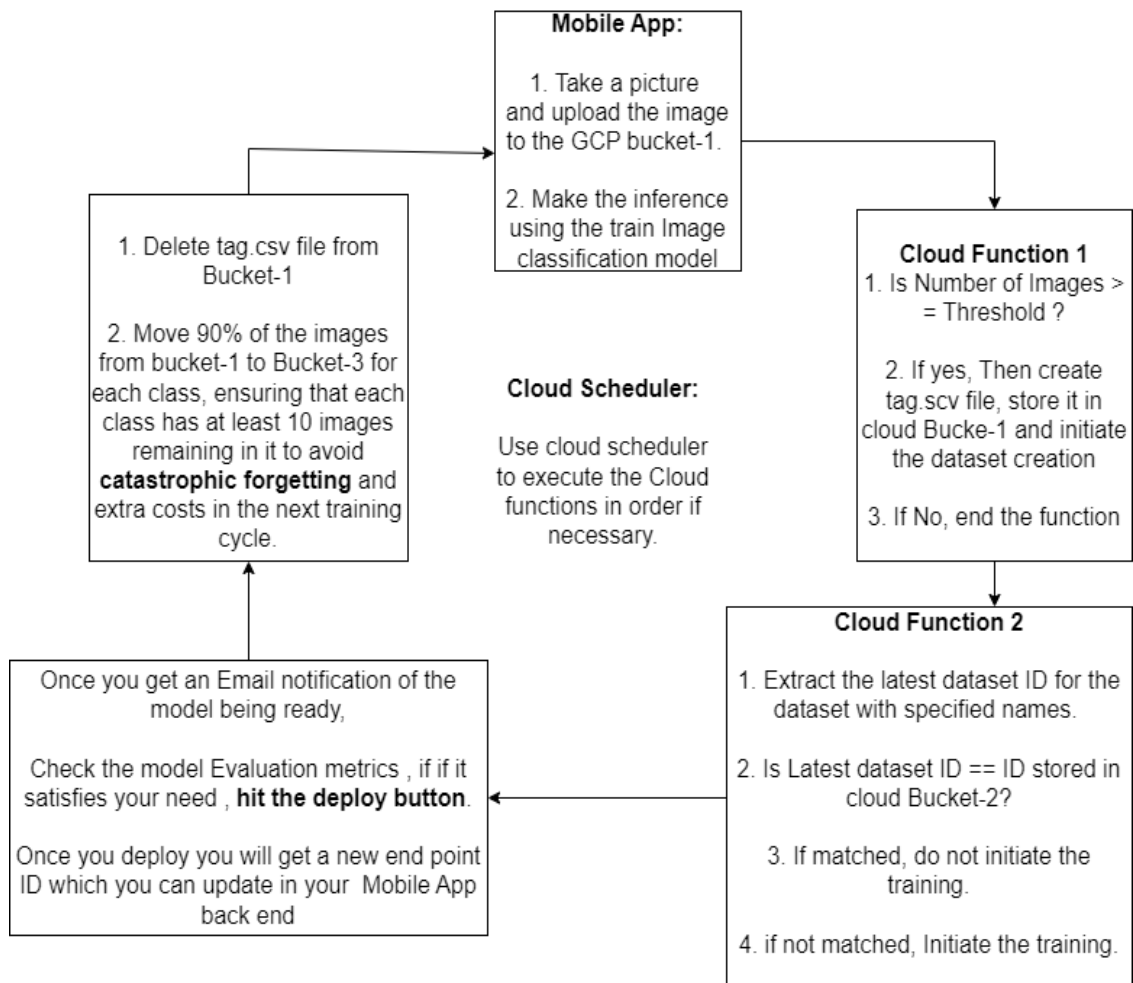
The core objective of this project is to craft a specialized image classification model tailored for a construction company, utilizing the cutting-edge AI services provided by Google Cloud Platform's Vertex AI. Once trained, this model will be incorporated into an intuitive mobile application, enabling users to capture images of materials on-site. The application will then identify these materials, assisting in the efficient sale or redistribution of unused or partially used construction products.

This initiative seeks to establish a seamless pipeline using cloud functions that span the entire workflow: from image collection and uploading to data preparation, model training, and deployment. Furthermore, the model will be integrated into the mobile application, streamlining the process of making accurate predictions in real-time. Designed to support continuous improvement through transfer learning, also known as incremental learning on GCP, the system aims to evolve constantly. As more images are collected, the model continuously updates and refines its learning, ensuring increased accuracy and relevance over time.

By addressing the pressing issue of construction waste through this AI-powered solution, the project not only contributes to environmental sustainability but also offers practical benefits to construction businesses by optimizing resource utilization and reducing unnecessary expenses.



### 3 System Design



Note: Initially before everything,

1. Create three cloud buckets: "construction-images" (Bucket-1), "bucket-to-store-dataset-id" (Bucket-2), and "backup" (Bucket-3) at the outset.
2. In Bucket-2, the file should be named "latest-dataset-id.txt"
3. Initially, if you have a pre-existing dataset named "dataset-construction" in your dataset store, store its ID in "latest-dataset-id.txt". Otherwise, leave the file empty.

Figure 1: System Diagram

## 4 Implementation

### 4.1 Step-by-step Description of the Pipeline Development

The important point we should not miss out while creating cloud function is by allocating more memory, we not only prevent out-of-memory errors but also potentially improve the function's performance. However, we need to remember that higher memory allocation can lead to higher costs, so it's a good practice to monitor the function's memory usage over time and adjust accordingly. If the function consistently uses far less memory than allocated, you might consider gradually reducing the memory size to find a balance between performance and cost. Before starting anything, first we need to create three Cloud-buckets.

**Cloud Bucket-1** to store the images and the tag.csv file later. **Cloud Bucket-2** with a .txt file inside it to store dataset ID. And **Cloud Bucket-3** to move 90 % of the images form cloud-bucket-1 after the end of each training process. The buckets have been named such only for naming convention for this report.

#### 4.1.1 [Pushing Images to GCP Bucket](#)

This Node.js script uses the Express framework to create a web server with functionality for uploading images to Google Cloud Storage (GCS). The images are uploaded to the specified GCP bucket in such a way that the image named "cement" is uploaded to the folder "CEMENT", the image named "brick" is uploaded to the folder named "BRICK", and so on.

This way, later on, these folder names can serve as class names while creating a dataset for training the model. Note that in GCP Cloud Storage, there are no actual “folders” in the traditional sense. For example, if you have a bucket named "my-bucket" and you create an object with the name "FOLDER1/file.txt", it appears as if there's a folder named "FOLDER1" containing a file named "file.txt". However, there isn't a separate entity called "FOLDER1"; it's just part of the object's name.

Filter by name prefix only ▼				
Filter Filter objects and folders				
<input type="checkbox"/>	Name	Size	Type	Created ?
<input type="checkbox"/>	AGGREGATE/	—	Folder	—
<input type="checkbox"/>	BRICK/	—	Folder	—
<input type="checkbox"/>	CEMENT/	—	Folder	—
<input type="checkbox"/>	METAL SHEET/	—	Folder	—
<input type="checkbox"/>	SAND/	—	Folder	—

Figure 2: Folders for each material being created in GCP bucket while pushing the images to the GCP bucket.

Here's a summary of its components and what each part does in the code.

1. Dependencies: The script imports necessary Node.js modules including `express` for server functionality, `@google-cloud/storage` for interacting with Google Cloud Storage, `path` for handling file paths, `fs` for file system operations, and `multer` for handling multipart/form-data, which is primarily used for uploading files.
2. Multer Configuration: Sets up Multer to handle file uploads. Files are saved in the 'uploads/' directory using their original names.
3. Express App Setup: Initializes an Express application and sets a port number for the server to listen on.
4. Google Cloud Storage Configuration: Establishes configuration for connecting to Google Cloud Storage, including the project ID, the path to the service account key file, and the name of the bucket where images will be uploaded.
5. Image Upload Function: Defines an asynchronous function `uploadImageToGCS` that uploads an image file to Google Cloud Storage. The function generates a unique file name for each image by appending a timestamp and converts the file name to uppercase. It organizes uploads into

folders based on the capitalized base name of the images. After uploading, the local copy of the image is deleted from the server.

6. File Upload Endpoint: Sets up a POST endpoint '/upload' where files can be uploaded through the server. The endpoint uses Multer to handle the file received in the form data, extracts necessary details (file path, original name, MIME type), and uses the `uploadImageToGCS` function to upload the file to Google Cloud Storage. Upon success, it sends a confirmation response; otherwise, it sends an error.

7. Homepage Endpoint: Sets up a GET endpoint '/' to serve an HTML file (assumed to be 'frontend.html') that likely contains the upload form, making it the landing page of the server.

8. Server Initialization: Starts the Express server to listen for requests on the specified port and logs a message to the console when the server is running.

Overall, this script is designed to provide a simple web interface for uploading images to a specified bucket in Google Cloud Storage, with each image being processed and organized based on its name and a unique timestamp before being uploaded.

#### 4.1.2 [Cloud Function – 1](#)

This Cloud Function aims to automate the creation of a dataset for image classification tasks in Google Cloud's Vertex AI. It monitors a specific Cloud Storage bucket for the presence of a sufficient number of images. If the number of images surpasses a predefined threshold, it generates a CSV file containing image paths and their corresponding labels. This CSV file is then uploaded to Cloud Storage and used to create a dataset in Vertex AI. The function operates asynchronously, returning a success message once the dataset creation process is initiated.

```

image_path,label
gs://construction-images/AGGREGATE/1696089490286.jpeg,AGGREGATE
gs://construction-images/AGGREGATE/1696089670674 (1).jpeg,AGGREGATE
gs://construction-images/AGGREGATE/1696089670674 (2).jpeg,AGGREGATE
gs://construction-images/AGGREGATE/1696089670674.jpeg,AGGREGATE
gs://construction-images/AGGREGATE/1696089670675 (1).jpeg,AGGREGATE
gs://construction-images/AGGREGATE/1696089670675.jpeg,AGGREGATE
gs://construction-images/AGGREGATE/1696089670676.jpeg,AGGREGATE
gs://construction-images/AGGREGATE/1696089670677.jpeg,AGGREGATE
gs://construction-images/AGGREGATE/1696089670680.jpeg,AGGREGATE

```

Figure 3: Snippet of tag.csv file.

General Information	
Last deployed	March 2, 2024 at 2:28:44 PM GMT+2
Region	us-central1
Memory allocated	1 GiB
CPU	583 millis
Timeout	120 seconds
Minimum instances	0
Maximum instances	100
Concurrency	1
Service account	<a href="mailto:518985248126-compute@developer.gserviceaccount.com">518985248126-compute@developer.gserviceaccount.com</a>
Build Worker Pools	—
Container build log	<a href="#">008c35cb-b42d-4ec1-afcb-a773f24c1d68</a>

Figure 4: Snippet of general information on cloud function – 1.

It's an HTTP-triggered function. You can increase or decrease the memory allocated, CPU, and timeout depending on the use case. For instance, the size of the tag.csv file being generated, the number of images in the bucket being queried, and other operations in the function itself can influence resource requirements.

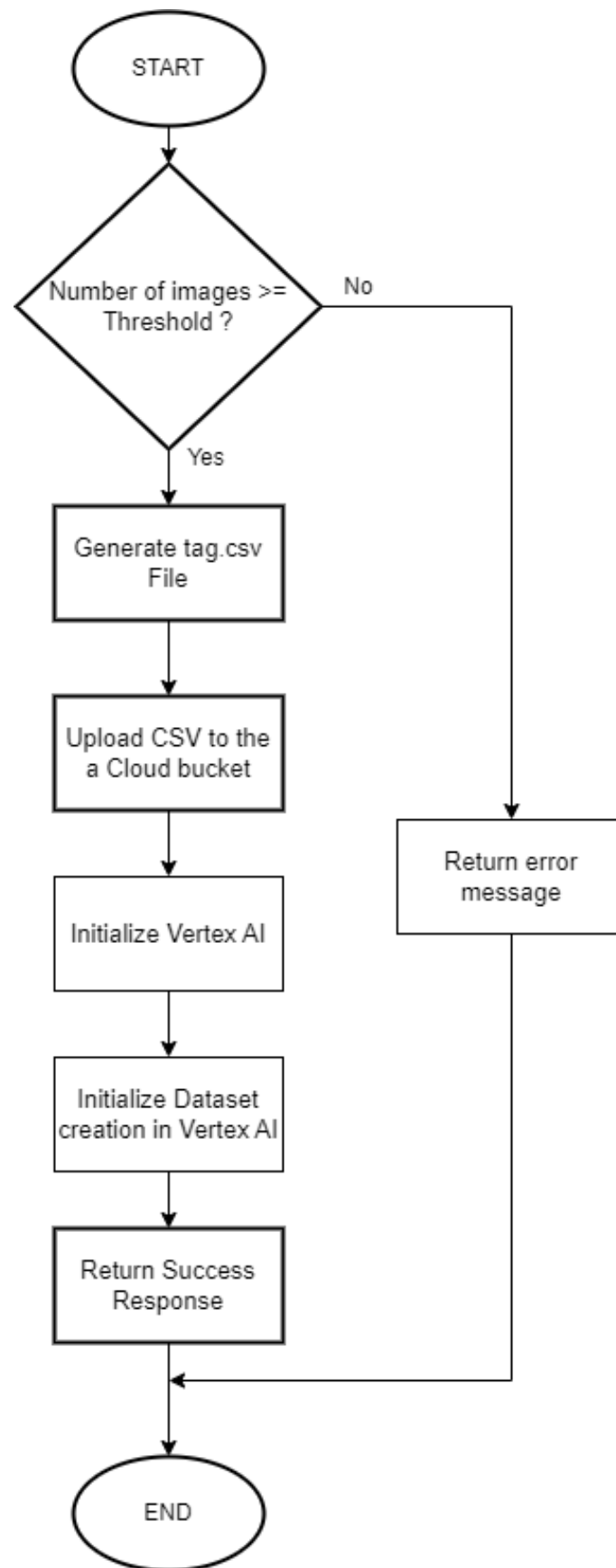


Figure 5: Flowchart for Cloud Function – 1.

### 4.1.3 [Cloud Function - 2](#)

This Cloud Function, automates the process of checking for new datasets in Vertex AI, comparing them with the latest dataset ID stored in Cloud Storage, and initiating model training if a new dataset is found. It first initializes the connection to Vertex AI, fetches the list of datasets sorted by creation time, and retrieves the latest dataset ID. Then, it checks the Cloud Storage bucket for the previously stored dataset ID. If the latest dataset ID differs from the stored one, it uploads the new ID to Cloud Storage and initiates model training using AutoML. If no new dataset is found, it logs a message indicating that the latest dataset has already been used for training.

General Information	
Last deployed	March 2, 2024 at 4:00:51 PM GMT+2
Region	us-central1
Memory allocated	1 GiB
CPU	583 millis
Timeout	200 seconds
Minimum instances	0
Maximum instances	100
Concurrency	1
Service account	<a href="mailto:518985248126-compute@developer.gserviceaccount.com">518985248126-compute@developer.gserviceaccount.com</a>
Build Worker Pools	—
Container build log	<a href="#">522f2240-5686-4b29-a406-e2efdc344bd8</a>

Figure 6: Snippet of General information on Cloud Function - 2

It's an HTTP-triggered function. You can increase or decrease the memory allocated, CPU, and timeout depending on the use case. For instance, the size of the time to extract the dataset from dataset section in GCP, creating a training job, and other such operations in the function itself can influence resource requirements.

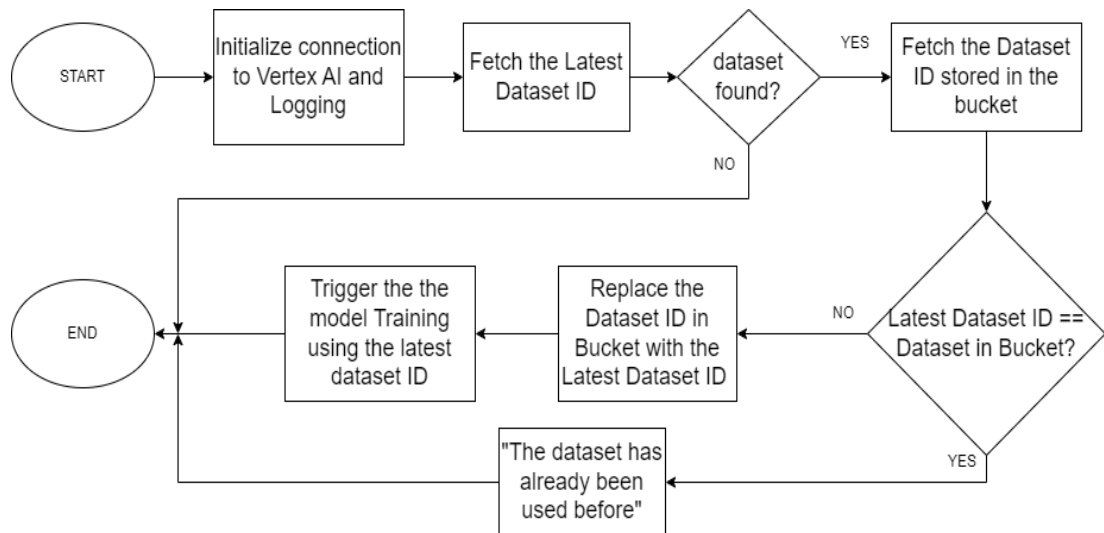


Figure 7: Flowchart for cloud function – 2.

#### 4.1.4 Deploying the Model

After the model is trained and an email notification is received, we can check the model evaluation metrics, such as accuracy, precision, recall, and the confusion matrix.

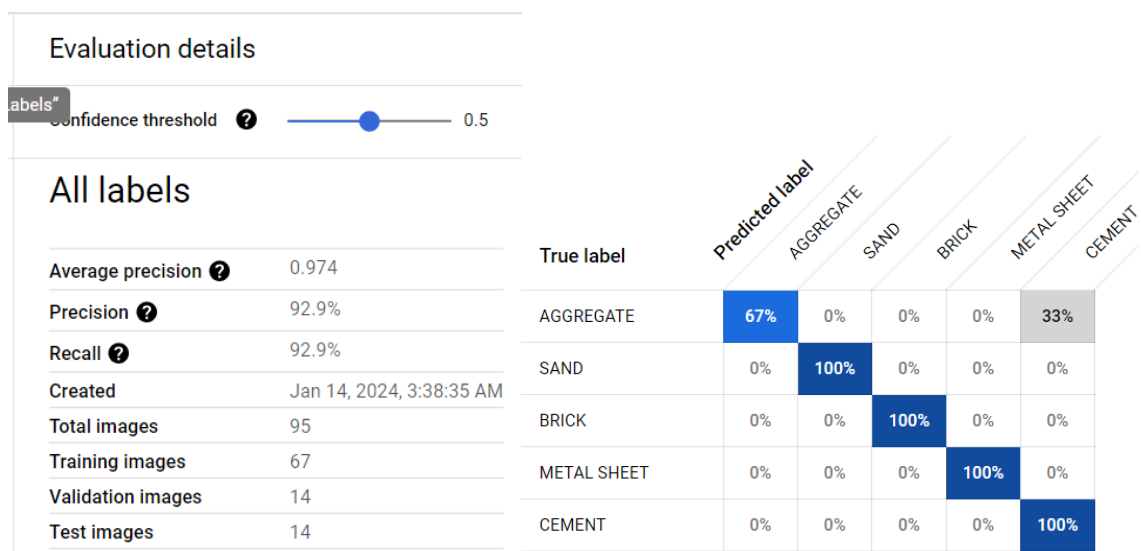


Figure 8: Evaluation Metrics for the Trained Model.




If the evaluation metrics meets the requirements, depending upon the use case, it can then be deployed by clicking the “Deploy” button.

EVALUATEDEPLOY & TESTBATCH PREDICTVERSION DETAILS

### Deploy your model

Endpoints are machine learning models made available for online prediction requests. Endpoints are useful for timely predictions from many users (for example, in response to an application request). You can also request batch predictions if you don't need immediate results.

DEPLOY TO ENDPOINT



Name	ID	Status	Models	Deployment resource pool	Region
No active endpoints containing this model					

Figure 9: Snippet of Button to deploy the model in Vertex AI.

#### 4.1.5 [Making Inferences](#)

Once the model is Deployed, GCP provides the Endpoint ID for the model to make the predictions.

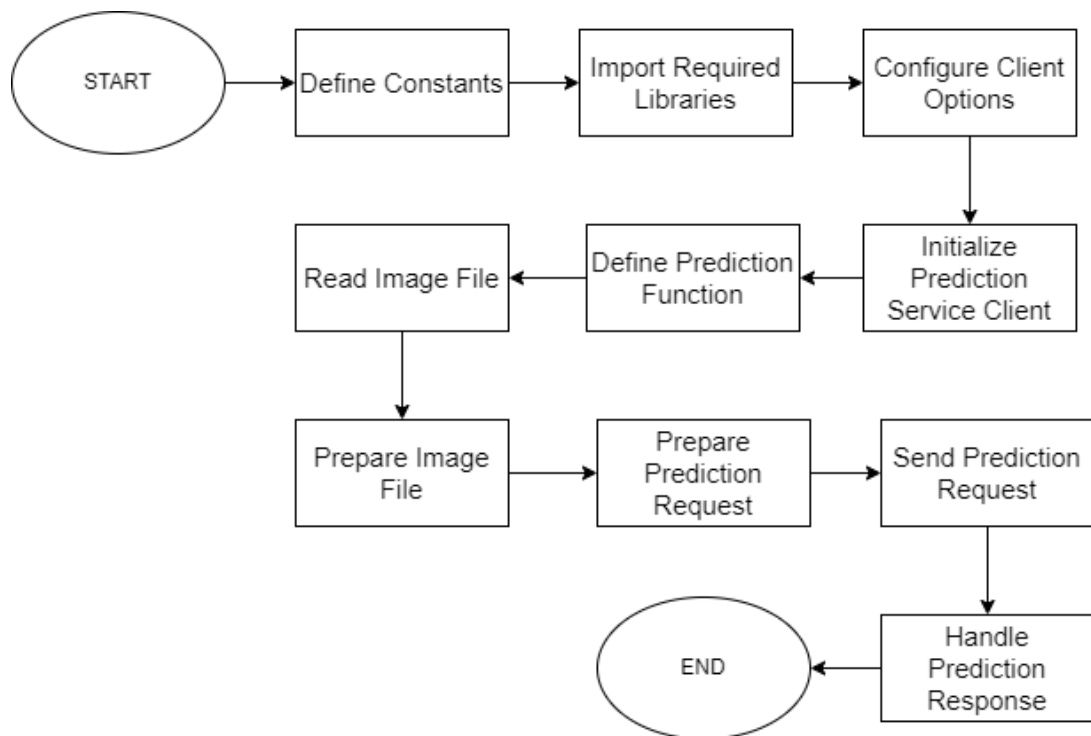


Figure 10: Flowchart for making inference.

#### 4.1.6 Moving 90 % of Images and Deleting tag.csv

**Moving 90% of the images from Cloud Bucket-1 to Cloud Bucket-3:** This means transferring the majority of the images stored in Cloud Bucket-1 to Cloud Bucket-3. This action is taken to prepare for Incremental Learning, a method where a model learns from new data while retaining knowledge from previous datasets.

**Leaving 10% of the images in Cloud Bucket-1 for the base model:** This portion of images is retained in Cloud Bucket-1 to serve as a base for the model. By keeping a subset of the original data, the model can maintain its understanding of previous patterns while adapting to new information.

**Concept of "catastrophic forgetting":** Catastrophic forgetting refers to the phenomenon where a neural network loses previously learned information when trained on new data. This issue can arise in transfer learning or incremental learning scenarios, where models need to adapt to new tasks or datasets without losing the knowledge gained from previous training.

**Efforts by vendors like Google to develop models with less catastrophic forgetting:** Google and other vendors are working on techniques and algorithms to mitigate catastrophic forgetting in neural networks. This involves designing models that can efficiently integrate new information while preserving knowledge from past experiences, thus enhancing their adaptability and performance in scenarios like incremental learning.

**Deleting tag.csv file:** It is wise to delete tag.csv before next round starts, so as to avoid creation of dataset using the same tag.csv file with cloud function-1 in next round.

## **5 Discussion and Cost Analysis.**

The overall goal of automating the collection of images in the cloud bucket into respective folders as different classes, using a cloud function to create the dataset using those images with the assistance of a tag.csv file, and training the model using a cloud function was achieved with the help of a combination of Cloud Scheduler and HTTP triggered Cloud Function.

The cost we need to focus on in this project is for the training and deployment processes.

### **5.1 Model Training Cost Analysis**

For training an image classification model, the price depends on the number of node-hours chosen. For example, the first model we trained on 4700 images

cost 8 node-hours. The node-hours per hour is shown in the figure below, provided by Google.

<a href="#">Image data</a> <a href="#">Video data</a> <a href="#">Tabular data</a> <a href="#">Text data</a>		
Operation	Price per node hour (classification)	Price per node hour (object detection)
Training	\$3.465	\$3.465
Training (Edge on-device model)	\$18.00	\$18.00
Deployment and online prediction	\$1.375	\$2.002
Batch prediction	\$2.222	\$2.222

Figure 11: Price list from GCP platform. [1]

Node hour budget

The maximum time, in compute hours, to spend training the model. More training time generally results in a more accurate model.

Note that training can be completed in less than the specified time if the system determines that the model is optimized and additional training would not improve accuracy. You are billed only for the hours actually used.

Typical training times	
Very small sets	1 hour
500 images	2 hours
1,000 images	3 hours
5,000 images	6 hours
10,000 images	7 hours
50,000 images	11 hours
100,000 images	13 hours
1,000,000 images	18 hours

Figure 12: Typical training times for fire-based Edge Model. [2]

So, the price for training 4700 images in our case was 8 nodes was:

Price for training =  $8 * 3.465 = \$27.72$

The training took 2.5 hours with 8 node hours to train 4700 images. The cloud seems to suggest the minimum node hours . It is not possible to go below the minimum node hours suggested by GCP while training cloud-based model.

However, during training, even when using as few as 108 images, the cloud-hosted model training did not allow training on fewer than 8 nodes. This is a bit inconvenient, as it means that even when training with a small number of images, such as 108, it still costs us 8 node hours, resulting in the same price.

Enter the **maximum** number of node hours you want to spend training your model.

You can train for as little as 8 node hours. You may also be eligible to train with free node hours. [Pricing guide](#)

The screenshot shows a web form for setting a training budget. It has a label "Budget \*" and a text input field containing the number "4". To the right of the input field is the label "Maximum node hours". Below the input field, there is a red warning icon and the text "Enter a number between 8 and 800". Below this, there is a toggle switch labeled "Enable early stopping" which is currently turned on (blue). To the right of the toggle is a description: "Ends model training when no more improvements can be made and refunds leftover training budget. If early stopping is disabled, training continues until the budget is exhausted."

Figure 13: GCP not allowing less than 8 nodes hours even just for 108 images.

## 5.2 Model Deployment Cost Analysis

As we can see from Figure 11, the deployment and online predictions cost is \$1.375 per node hour. One thing to note is that, while deploying the model endpoint, GCP asks to reserve at least one node hour. This means it will charge for that one node hour continuously as long as the model stays deployed. For instance, when we deployed our first model and left it deployed for 12 hours, it had cost us  $12 * 1.375 = \$16.5$ .

It is not advisable to leave the cloud-hosted model endpoint deployed, as it will continuously incur charges for at least one node for each hour, regardless of whether predictions are being made or not.

In terms of pricing, the **Edge Model** for end devices offers a more cost-effective solution compared to the **cloud-based model**. Training the Edge Model costs \$18 per node hour, while the cloud-based model is priced at \$3.465 per hour. However, the Edge Model presents a significant advantage: it can be downloaded and utilized offline on mobile devices for predictions without the need for deployment as a model endpoint. Conversely, while the cloud-based model's predictions on cloud resources are more costly, they offer greater accessibility and scalability.

Edge model	Cloud model
<ol style="list-style-type: none"> <li>1. Training the Edge Model costs \$18 per node hour.</li> <li>2. The Minimum nodes allowed for training starts from 1 Nodes.</li> <li>3. The model can be downloaded as <b>TF lite version</b> for mobile devices, <b>or TF Saved</b> to run on a Docker container or <b>TensorFlow.js</b> package model to run on browser and in Node.js. Or it can also be <b>Deployed to the Endpoint</b>.</li> </ol>	<ol style="list-style-type: none"> <li>1. Cloud-based model training is priced at \$3.465 per node hour.</li> <li>2. Minimum node allowed for training starts from 8 Nodes.</li> <li>3. The model can only be <b>deployed to Endpoint</b>. However Cloud-based Model offers service of incremental learning while edge model does not..</li> </ol>

<p>4. The prediction can be made offline using mobile resources, incurring no costs for prediction. However, if you wish, you can also make predictions online using cloud resources, which will incur costs as shown in Figure 11.</p> <p>5. Since it has an option to download the model and can be used offline, there is no cost for making predictions, and there is no need to reserve the node if it is not deployed to the endpoint to make online inferences.</p>	<p>4. The prediction can be made online only, incurring a small amount of cost for each prediction as shown in Figure 11.</p> <p>5. Since it has an option to download the model and can be used offline, there is no cost for making predictions, and there is no need to reserve the node if it is not deployed to the endpoint to make online inferences.</p>
--	--

## 5.3 Cloud Function Costs

### 5.3.1 Cloud Function Invocations Costs

#### Invocations

Function invocations are charged at a flat rate regardless of the source of the invocation. This includes [HTTP function](#) invocations from HTTP requests, events forwarded to [background](#) or [CloudEvent](#) functions, and invocations resulting from the [call API](#). The pricing tiers shown below are based on the total number of function invocations across *all* functions associated with a particular Google Cloud Platform [billing account](#).

Invocations per month	Price/million
First 2 million	Free
Beyond 2 million	\$0.40

Figure 14: Cloud Function Invocations Cost. [3]

In our case, the **cloud functions** are invoked once for instance, **every three months** using cloud Scheduler, so the costs won't be significant, especially considering up to 2 million invocations, if not for deployment and storage costs.

### 5.3.2 Cloud Function Deployment Cost:

Cloud Functions (2nd gen) exclusively uses Artifact Registry for storing functions.

If functions are stored in Container Registry, small charges may occur after deployment due to storage costs, approximately \$0.026 per GB per month for multiregional storage.

For example, deploying fifteen 1st gen Node.js 10 functions might result in charges of around \$0.03 per month for storage.

If functions are stored in Artifact Registry, charges occur only if you exceed the free tier of storage.



Each function's container is stored in an image registry until deletion, resulting in a small monthly charge (unless using Artifact Registry and staying within the free tier limit). [[4](#)]

## 6 Results

- The project successfully implemented automation using cloud functions for executing tasks periodically, such creating tag.csv file , using it for dataset preparation and model training.
- Cloud Function 1 was scheduled to trigger monthly (or every three months), followed by Cloud Function 2 on the subsequent day, streamlining the process.
- Evaluation metrics is manually reviewed before model deployment to ensure reliability and meet requirements.

## 7 Conclusions

- While automation proved effective for certain tasks, limitations were encountered in using cloud functions alone, particularly in retrieving previous base model IDs for training with new data.
- Extensive research and trials revealed that not all functionalities achievable through the GUI interface of GCP could be replicated using cloud functions.
- In cases where automation is not imperative, utilizing the GUI interface offers greater control over dataset creation, model training, and decision-making processes.
- Future considerations include the need for manual intervention, especially in critical stages like evaluating model performance before deployment. This ensures reliability and minimizes potential issues, ultimately saving time and resources.
- Considering the fact that one has to reserve and pay for at least one node hour continuously if the model endpoint is deployed, we can conclude that the Edge model is better in this perspective. However, it ultimately depends on the use case. If the use case demands very high precision and economy is not an issue, then the cloud-based model would be more suitable.

## References

- 1 [https://cloud.google.com/vertex-ai/pricing?\\_gl=1\\*1b8e3su\\*\\_ga\\*MjA0OTkwOTU4NS4xNzA0OTgwOTg0\\*\\_ga\\_WH2QY8WWF5\\*MTcwOTQxOTU3My4zNi4xLjE3MDk0MjE2NDluMC4wLjA.&\\_ga=2.140708657.-2049909585.1704980984](https://cloud.google.com/vertex-ai/pricing?_gl=1*1b8e3su*_ga*MjA0OTkwOTU4NS4xNzA0OTgwOTg0*_ga_WH2QY8WWF5*MTcwOTQxOTU3My4zNi4xLjE3MDk0MjE2NDluMC4wLjA.&_ga=2.140708657.-2049909585.1704980984)
- 2 <https://firebase.google.com/docs/ml/train-object-detector>
- 3 <https://cloud.google.com/functions/pricing>
- 4 [https://cloud.google.com/functions/pricing#deployment\\_costs](https://cloud.google.com/functions/pricing#deployment_costs)