Overview

Einstein Vision helps to recognize and classify images or objects at scale using deep learning model. Einstein Vision supports the following APIs such as Einstein Image Classification & Einstein Object Detection.

Terminologies

Dataset– Training data or sample set of master data

Label– Segregation of similar types of data using Label

Model– A machine-learning program to classify images or objects with respect to label

Training– Process to train the dataset
Prediction– Result to predict the data with respect to trained dataset using the specified model

Setup

Signup on Einstein Platform Services Account- https://api.einstein.ai/signup

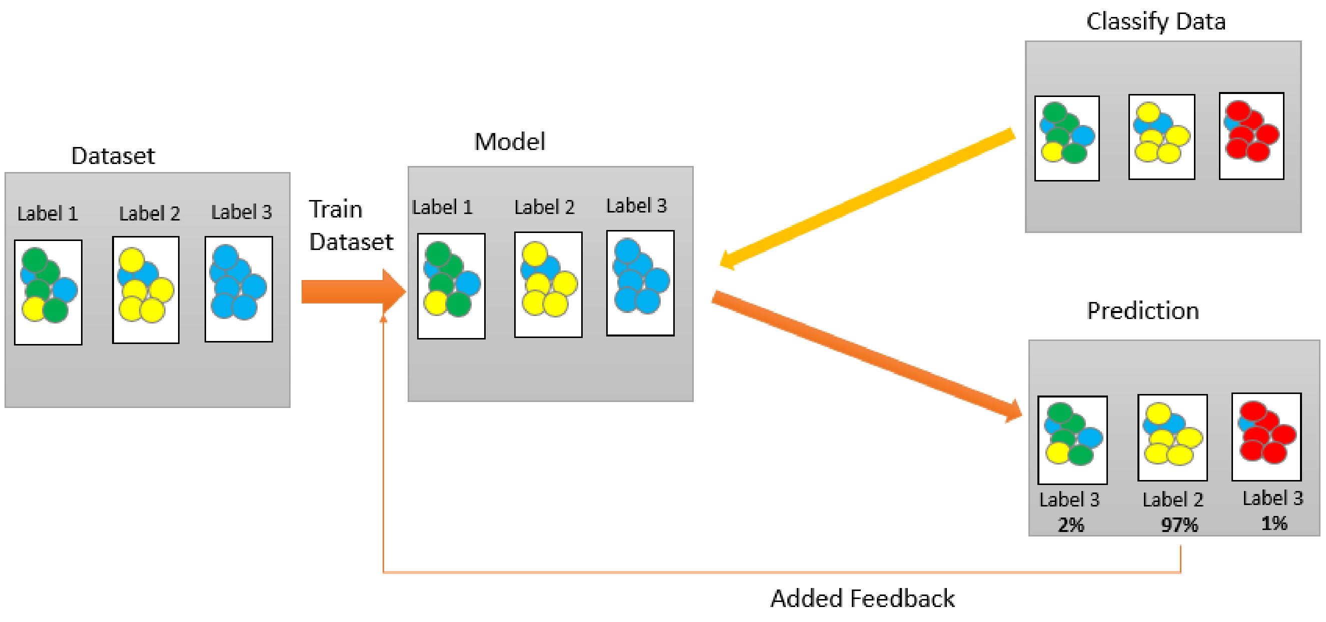
On the activation page, click "**Download Key**" to save the key file named "**einstein_platform.pem**" locally.

Install cURL- https://curl.haxx.se/download.html

Get Token- https://api.einstein.ai/token

Click on "**Get Token**" button and Keep the access token to execute the next operations.

Process flow (Create Dataset with Labels -> Train the Dataset -> Generate the Model -> Classify the data -> Prediction Score -> Add feedback)



Einstein Image Classification

Enables developers to recognize and classify images at scale using deep learning models with the help of Einstein Image Classification API.

Create the Dataset

Organize the same types of images in each folder with correct label and create a zip file. For example, that we can segregate the quality of images of soils such as Good soil, Bad soil & Average soil in each folder and make a zip file named "Soils.zip". Einstein Vision will predict the quality of soil for farming.

***[Note:** Please replace the <TOKEN>, <DATSET_ID>, <MODEL_ID> with the proper values for the following syntaxes.]*

Upload the Dataset

```
curl -X POST -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" -H "Content-Type: mu
ltipart/form-data" -F "type=image" -F "data=@D:\Einstein Vision\Soils\Soils.zip" https://api.eins
tein.ai/v2/vision/datasets/upload
```

Replace TOKEN from https://api.einstein.ai/token | **data** is being used for local drive address to get the dataset

For the web address, we need to use path instead of using data.
"path=https://einstein.ai/images/mountainvsbeach.zip"

Keep the generated Dataset Id.

This is the asynchronous process. In this case, we need to check the upload status by the following cURL command

```
curl -X GET -H "Authorization: Bearer <TOKEN>" \
            -H "Cache-Control: no-cache" \
            https://api.einstein.ai/v2/vision/datasets/<DATASET_ID>
```

We can upload the data in synchronous way by using upload/sync in the main data uploading **cURL** com mand <https://api.einstein.ai/v2/vision/datasets/upload**/sync**>

Train the Dataset

```
curl -X POST -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" -H "Content-Type: mu
ltipart/form-data" -F "name= Soils Model" -F "datasetId=<DATASET_ID>" https://api.einstein.ai/v2/
vision/train
```

Need to set the Dataset Id from the previous step.
Keep the generated Model Id.

We can provide the name of the model as per business model.

python -m json.tool can be used to set the output as in JSON format in command prompt in this way

```
curl -X POST -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" -H "Content-Type: mu
ltipart/form-data" -F "name= Soils Model" -F "datasetId=<DATASET_ID>" https://api.einstein.ai/v2/
vision/train|python -m json.tool
```

To get the status of training

```
curl -X GET -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache"
https://api.einstein.ai/v2/vision/train/<YOUR_MODEL_ID>
```

After completion of training, it will return progress value as **1** with the status as "**SUCCEEDED**".

Training Quality Check

To get accuracy, f1 score and confusion matrix

```
curl -X GET -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" https://api.einstein.
ai/v2/vision/models/<MODEL_ID>
```

Need to set the Model Id from the previous step.

To get Learning Curve about the training

```
curl -X GET -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" https://api.einstein.
ai/v2/vision/models/<MODEL_ID>/lc
```

Get the Predicted Result

```
curl -X POST -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" -H "Content-Type: mu
ltipart/form-data" -F "sampleContent=@D:\Einstein Vision\Soils\Good\good1.jpg" -F "modelId=<MODEL
_ID>" https://api.einstein.ai/v2/vision/predict
```

sampleContent is used to refer the path of the image in the local drive.
samleLocation can be used here if we want to take the image dirrectly from http.

Add Feedback

```
curl -X POST -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" -H "Content-Type: mu
ltipart/form-data" -F "modelId=<MODEL_ID>" -F "data=@D:\Einstein Vision\Soils\Bad\good16.jpg" -F
"expectedLabel=Good" https://api.einstein.ai/v2/vision/feedback
```

Retraining

```
curl -X POST -H "Authorization: Bearer <TOKEN>" –H "Cache-Control: no-cache" –H "Content-
Type: multipart/form-data" -F "modelId=<MODEL_ID>" -F "trainParams={\"withFeedback\" : true}" htt
ps://api.einstein.ai/v2/vision/retrain
```

Create a new Model with the feedbacks

If we want, we can also create a new model with the latest feedbacks.

```
curl -X POST -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" -H "Content-
Type: multipart/form-data" -F "name=Soils Model With Feedback" -F "datasetId=<DATASET_ID>
" -F "trainParams={\"withFeedback\" : true}" https://api.einstein.ai/v2/vision/train
```

Delete the Dataset

```
curl -X DELETE -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" https://ap
i.einstein.ai/v2/vision/datasets/<DATASET_ID>
```

The deletion may not occur immediately.

Keep the deleted object id.

to get the status of deletion

```
curl -X GET -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" https://api.e
instein.ai/v2/vision/deletion/<DELETEDOBJECT_ID>
```

Delete the Model

```
curl -X DELETE -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" -H "Conten
t-Type: multipart/form-data" https://api.einstein.ai/v2/language/models/<MODEL_ID>
```

Einstein Vision Result by Apex Class

1. Upload the private key file named '**einstein_platform.pem**' into Salesforce Files.
2. Create Remote Site with the following URL: https://api.einstein.ai
3. Clone or Download a ZIP file for JWT Master. https://github.com/salesforceidentity/jwt
4. Create **JWT & JWTBearerFlow** Apex Classes to generate access token.
5. Clone or Download the Apex Classes from https://github.com/MetaMind/apex-utils
6. Create Apex Util Classes such as HTTPFormBuilder & Vision.
7. To see the output of prediction, we can take a sample of Visualforce Page provided by Salesforce. Please create the Visualforce Page and respective Apex Class from README.md file of apex-utils GIT repository.
8. We can get the output by executing a trigger also. We can write a trigger on **ContentVersion** object to get the predicted result when user uploads any image in a file for a record. Remember, that trigger should be in asynchronous nature.

Einstein Object Detection

Enables developers to recognize the object details like object coordinates, size, location of each object as well as developers can distinguish multiple distinct objects using deep learning models with the help of Einstein Object Detection API.

Create the Dataset

Organize the in each folder with correct label and store all the necessary information like height, coordinates, name, image uRL etc. We can refer the object detection zip file provided by Salesforce. The object detection zip file must contain the images and an annotations.csv file in the required format like:

https://einstein.ai/images/alpine.zip

Upload the Dataset

```
curl -X POST -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" -H "Content-
Type: multipart/form-data" -F "path=https://einstein.ai/images/alpine.zip" -F "type=image
-detection" https://api.einstein.ai/v2/vision/datasets/upload
```

Please notice that to upload the data for Einstein Object Detection, we need to write type=”image-detection”.

Train the Dataset

Same as Einstein Image Classification

Get the Predicted Result

```
curl -X POST -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" -H "Content-
Type: multipart/form-data" -F "sampleContent=@C:\data\alpine.jpg" -F "modelId=<YOUR_MODEL
_ID>" https://api.einstein.ai/v2/vision/detect
```

Here we have used “**detect**” keyword instead of “predict” which has been used to get the predicted result for Image Classification.

Prebuilt Models

We can use some prebuilt models provided by Einstein Vision to get the predicted result.

- Food Image Model – To get the result we must use modelId as “**FoodImageClassifier**” like:

```
curl -X POST -H "Authorization: Bearer <TOKEN>" -H "Cache-Control: no-cache" -H "Content-Type
: multipart/form-data" -F "sampleLocation=http://einstein.ai/images/foodimage.jpg" -F "modelI
d=FoodImageClassifier" https://api.einstein.ai/v2/vision/predict
```

- General Image Model - To get the result we must use modelId as "**GeneralImageClassifier**"
- Scene Image Model - To get the result we must use modelId as "**SceneClassifier**"
- Multi-Label Image Model -To get the result we must use modelId as "**MultiLabelImageClassifier**"

Training Data & Test Data

- Training data—Samples are being used by the training process to create the model.
- Test data— Samples are being used by the training process to test the model accuracy.

Einstein Vision—90% of the data/samples is used to create the model, and 10% is used to test the model accuracy.

Using Global Dataset

We can use Global Dataset to create negative class in our model. Let's say, we have created three Soil labels such as Good, Average & Bad. But it will be good if we use global dataset to create the separate negative class, called "Other" so images not like soil would be returned as "Other" images. Please follow this link: https://metamind.readme.io/docs/use-global-datasets

Best Practices

- Try to keep at least 1,000 examples per dataset label to get the better prediction.
- Objects in the images should be visible and recognizable.
- Please keep the images in forward-facing and not in an angle.
- Try to keep each dataset label should have same number of images as well as variety of images with the following characteristics
  - Color
  - Black and white
  - Blurred
  - With other objects the object might typically be seen with
  - With text and without text (if applicable)
- In a binary dataset, include images in the negative label that look like images in the positive label also to get the correct probability of segregation.
- For a multi-label model, try to include images with objects that appear in different areas within the image.

References

- Introduction to Salesforce Einstein Vision: https://metamind.readme.io/docs/introduction-to-the-einstein-predictive-vision-service
- Trailhead: Quick Start: Einstein Image Classification : https://trailhead.salesforce.com/en/content/learn/projects/predictive_vision_apex

Author: Santanu Pal, 14x Salesforce Certified System Architect & Application Architect
Blog: https://www.santanuatonline.com