

IMAGE RECOGNITION WITH IBM CLOUD VISUAL RECOGNITION

INTRODUCTION:

In this project, we aim to develop a image recognition platform by utilizing the advanced capabilities provided by IBM Cloud Visual recognition.

The primary objective of this project is to create a image recognition system capable of classifying objects, scenes, or patterns present in images accurately.

PROBLEM STATEMENT:

The problem in cloud application development for image recognition with IBM involves creating a robust and efficient system for analyzing and processing images using IBM's cloud services. This encompasses various challenges, including:

1. Scalability:

Ensuring the application can handle a large volume of image data efficiently as the user base grows.

2. Accuracy:

Achieving high accuracy in image recognition tasks to meet user expectations.

3. Cost Optimization:

Managing the cost associated with cloud resources, as image processing can be resource-intensive.

4.Security:

Protecting sensitive image data and ensuring compliance with data privacy regulations.

SOLUTION APPROACH:

To address the problem statement and achieve our objectives, we will follow a systematic approach that encompasses the following key steps:

1.Choose IBM Cloud Services:

Select the appropriate IBM cloud services for image recognition. IBM Watson Visual Recognition is a suitable option, offering pre-trained models and customization capabilities.

2.Data Collection and Preprocessing:

Gather a diverse dataset of images relevant to your application. Preprocess the images, ensuring they are in the right format and resolution for analysis.

3.Model Training and Customization:

Train the image recognition model using IBM Watson Visual Recognition. Fine-tune the model to improve accuracy for your specific use case. This might involve creating custom classifiers.

4. Scalability Planning:

Design your application architecture to be scalable. Utilize cloud services like IBM Cloud Functions or Kubernetes for auto-scaling based on traffic.

5. Cost Monitoring and Optimization:

Implement cost-monitoring tools to keep track of resource usage and optimize costs. This may involve using serverless computing to reduce idle resource expenses.

6. Security Measures:

Implement security measures to protect image data. Use encryption, access control, and compliance with regulations like GDPR if handling personal images.

7. User Interface Development:

Create a user-friendly interface for users to interact with the image recognition system. This may involve developing a web or mobile app that integrates with IBM's APIs.

8. Testing and Quality Assurance:

Thoroughly test the application for accuracy, scalability, and security. Use testing frameworks and perform load testing to ensure it can handle peak loads.

9. Deployment and Monitoring:

Deploy the application on the IBM Cloud. Implement monitoring tools to track system performance, detect anomalies, and troubleshoot issues promptly.

10. User Education and Support:

Provide user documentation and support to help users effectively utilize the image recognition capabilities of the application.

11. Feedback Loop:

Establish a feedback loop to continuously improve the model's accuracy and the application's performance based on user feedback and changing requirements.

12. Regular Updates:

Keep the application and underlying services up to date with the latest enhancements and security patches provided by IBM.

Project Plan: Emotion-Enriched Image Caption Generator

Table of Contents

1. Project Overview

- Introduction

- Objectives

- Target Audience

2. Technologies and Tools

- Image Recognition Model

- Sentiment Analysis Model

- Programming Languages and Libraries

3. Data Collection and Preprocessing

- Image Dataset

- Emotion Annotation

- Data Preprocessing

4. Model Development

- Image Recognition Model

- Sentiment Analysis Model

5. Integration

- Combining Image and Sentiment Analysis
- Caption Generation

6. User Interface

- Web Application
- User Interaction Flow

7. Testing and Evaluation

- Model Evaluation
- User Testing

8. Deployment

- Hosting the Application

- API Endpoints

9. Documentation and User Guide

- Usage Instructions
- Documentation of APIs

10. Future Improvements

- Potential Enhancements
- Scaling Strategies

11. Conclusion

- Project Recap
- Achievements and Lessons Learned

1. Project Overview

Introduction

This project aims to develop an Image Caption Generator that not only describes the content of images but also captures the emotions and mood portrayed in those images using sentiment analysis. The system will be capable of providing rich and emotionally expressive captions for a wide range of images.

Objectives

- Develop an image recognition model to identify objects and scenes in images.
- Create a sentiment analysis model to assess the emotional tone of images.
- Combine image recognition and sentiment analysis to generate emotion-enriched captions.

Target Audience

- Content creators
- Social media users
- Anyone seeking creative and expressive image captions

2. Technologies and Tools

- Image Recognition Model: Utilize deep learning frameworks like TensorFlow or PyTorch.

- Sentiment Analysis Model: Choose a suitable sentiment analysis library or pre-trained model.

- Programming Languages and Libraries: Python, Flask (for web interface), and relevant deep learning libraries.

3. Data Collection and Preprocessing

Image Dataset

- Collect a diverse dataset of images that cover various themes, objects, and emotions.

- Ensure proper permissions for image usage.

Emotion Annotation

- Annotate the images with emotion labels that represent the emotions conveyed by the images.

Data Preprocessing

- Resize and normalize images.

- Prepare emotion labels for model training.

4. Model Development

Image Recognition Model

- Develop a deep learning model (e.g., CNN) for image recognition.

- Train the model using the annotated image dataset.

Sentiment Analysis Model

- Create or fine-tune a sentiment analysis model using pre-trained models like BERT or VADER.
- Train the model using emotion-labeled images.

5. Integration

Combining Image and Sentiment Analysis

- Combine the output of the image recognition model and the sentiment analysis model to understand the image's content and emotions.

Caption Generation

- Develop algorithms that generate captions by considering both the image content and emotional analysis.

6. User Interface

Web Application

- Create a user-friendly web application where users can upload images.
- Implement an interface for users to receive emotion-enriched captions.

User Interaction Flow

- Define how users will interact with the application.

- Specify the input and output flow.

7. Testing and Evaluation

Model Evaluation

- Assess the accuracy of the image recognition and sentiment analysis models.

- Evaluate the quality of generated captions.

User Testing

- Conduct user testing to gather feedback on the application's usability and the quality of generated captions.

8. Deployment

Hosting the Application

- Deploy the application on a suitable hosting platform (e.g., AWS, Heroku).

API Endpoints

- Create RESTful API endpoints for image upload and caption retrieval.

9. Documentation and User Guide

Usage Instructions

- Document how to use the web application and its features.
- Provide clear instructions for users.

Documentation of APIs

- Document the API endpoints and their usage for developers.

10. Future Improvements

Potential Enhancements

- Discuss potential enhancements such as real-time image processing and support for multiple languages.

Scaling Strategies

- Consider strategies for scaling the application based on user demand.

11. Conclusion

Project Recap

- Summarize the project's goals and achievements.

Achievements and Lessons Learned

- Reflect on the project's successes and the lessons learned during its development.

How to Build Your First Image

Recognition Classifier with IBM Watson

Visual Recognition


Before cracking on with actually building a classifier the first thing you'll need is an IBMid. Why?

Well, you're going to be using the Watson Visual Recognition API to train and test your classifier. These credentials allow you to access the free tier of IBM AI products through the IBM Cloud platform (side note: this was previously called IBM Bluemix).

If you don't have an IBMid already head on over to the link below and sign up for one. Once you've got an account setup, take note of your credentials as you'll need them in the next step!

Create your IBM account

Access to trials, demos, starter kits, services, and APIs.



Sign up for an IBMid

Already have an IBM account? [Log in](#)

Email *

First name *

Last name *

Country or region * (?) ✓

Australia ▼

Set a password *

SHOW

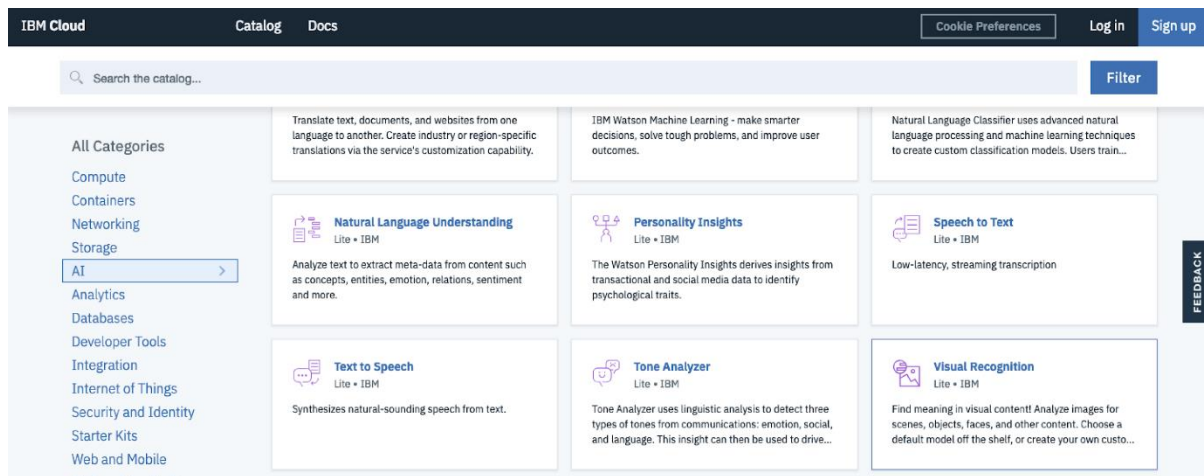
- 8 characters minimum
- One uppercase character
- One lowercase character
- One number

STEP 2 – CREATE A SERVICE

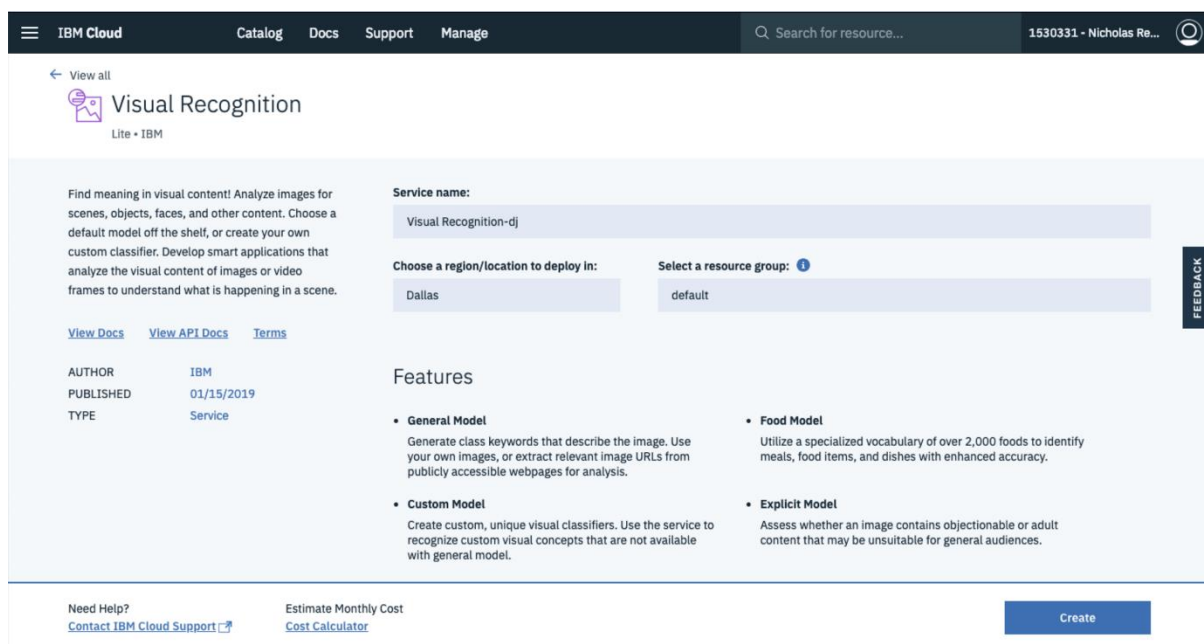
The next step is to spin up a service on IBM Cloud which will allow you to access the Visual Recognition API. This can all be done through the IBM Cloud console. Start by heading over to the

IBM Cloud Catalog

From there, go to AI



Here you'll be asked to name your service, select a region and select a resource group. These can be left with their defaults.



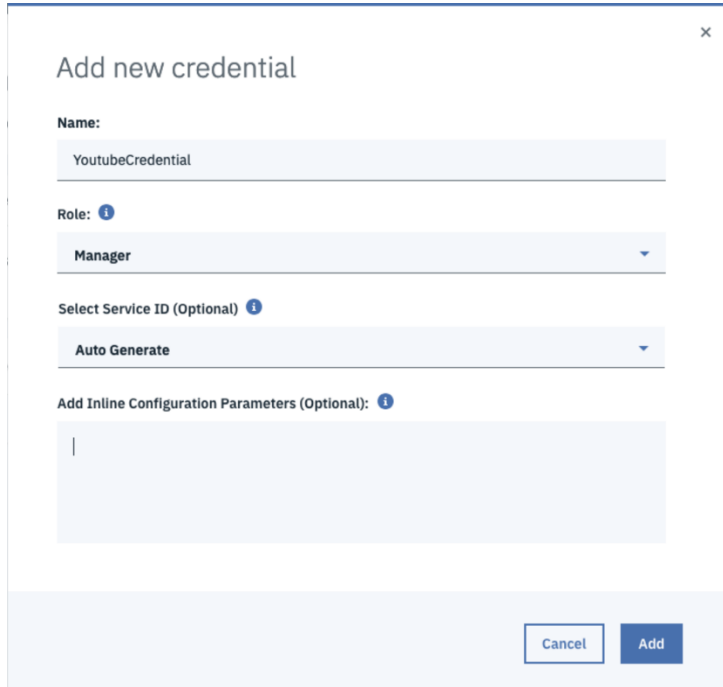
If you scroll down you'll be asked to select a plan. The Lite plan is more than sufficient when just starting out. Select that one and click **Create**

Pricing Plans			Monthly prices shown are for country or region: Australia
PLAN	FEATURES	PRICING	
✓ Lite	1,000 Events per month towards: Pre-trained model classification (General, Face, Food, Explicit) (images) Custom Model classification (images) Custom Model training (images) 2 Custom Models 1 Lite Plan instance per IBM Cloud Organization Free Exports to Core ML	Free	
<p>The Lite Plan gets you started with 1,000 events (images) per month and the ability to train two Custom Models. Users wishing to use more premium features or increase usage must upgrade to a Standard Plan or a Subscription Plan.</p> <p>Lite plan services are deleted after 30 days of inactivity.</p>			

STEP 3 – GENERATE API CREDENTIALS

The Visual Recognition API uses token-based [IAM to authenticate requests](#). The IAM credentials can be generated once you've created your service by going to Service Credential then Selecting New Credential.

Once there, create a new credential by entering a name, changing the role to Manager and selecting Auto Generate (optional) for the Select Service ID then hitting Add.



Add new credential

Name: YoutubeCredential

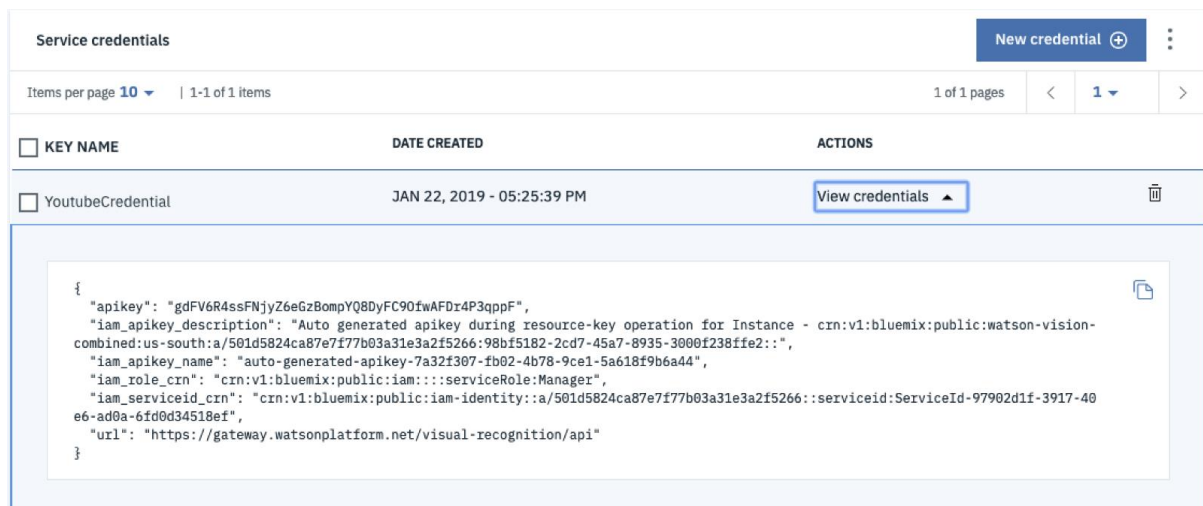
Role: Manager

Select Service ID (Optional): Auto Generate

Add Inline Configuration Parameters (Optional):

Cancel Add

This will generate new credentials which will allow you to use the API when accessing it using Python later on. Click View Credentials and copy the apikey, you'll need it in Step 5.



KEY NAME	DATE CREATED	ACTIONS
YoutubeCredential	JAN 22, 2019 - 05:25:39 PM	View credentials

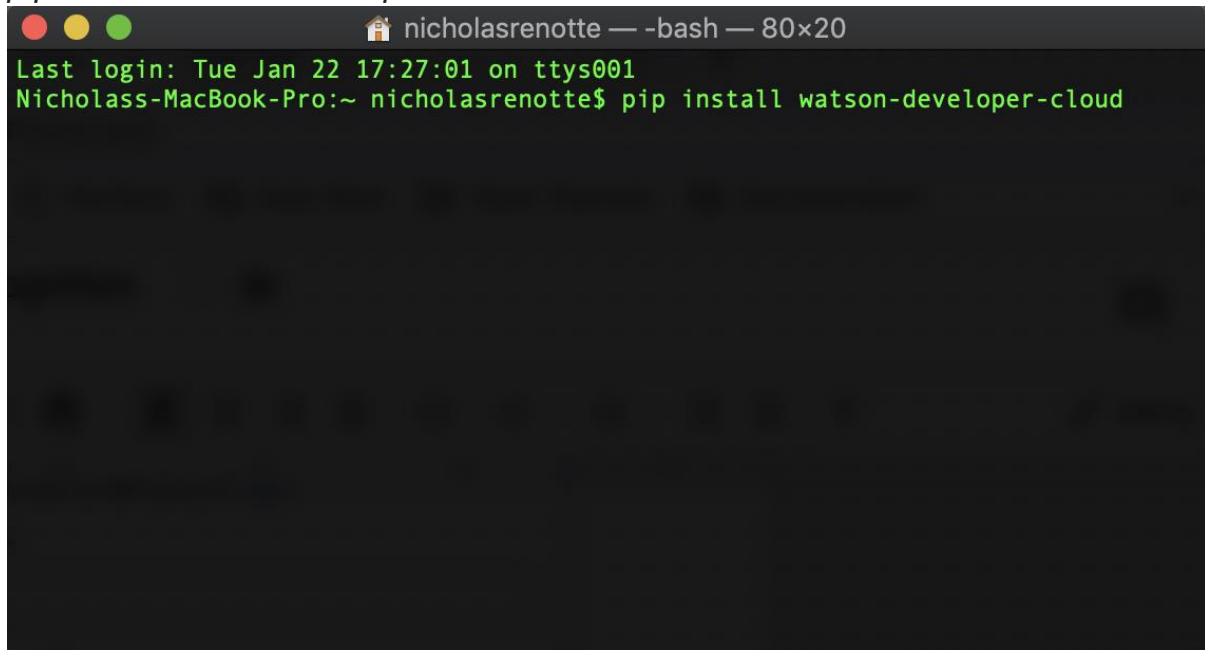
```
{  "apikey": "gdFV6R4ssFNjyZ6eGzBompYQ8DyFC90fwAFDz4P3qppF",  "iam_apikey_description": "Auto generated apikey during resource-key operation for Instance - crn:v1:bluemix:public:watson-vision-combined:us-south:a/501d5824ca87e7f77b03a31e3a2f5266:98bf5182-2cd7-45a7-8935-3000f238ffe2::",  "iam_apikey_name": "auto-generated-apikey-7a32f307-fb02-4b78-9ce1-5a618f9b6a44",  "iam_role_crn": "crn:v1:bluemix:public:iam::::serviceRole:Manager",  "iam_serviceid_crn": "crn:v1:bluemix:public:iam-identity::a/501d5824ca87e7f77b03a31e3a2f5266::serviceid:ServiceId-97902d1f-3917-40e6-ad0a-6fd0d34518ef",  "url": "https://gateway.watsonplatform.net/visual-recognition/api"}
```

STEP 4 – INSTALL WATSON DEVELOPER CLOUD

We're out of setup world now and we're finally delving into some code. Because this example is going to use Python to access the visual recognition service, we're going to need some modules to help us out.

The key module that we'll be using is `watson-developer-cloud`. To install it run:

pip install watson-developer-cloud

A screenshot of a macOS terminal window. The title bar shows three colored window control buttons (red, yellow, green) on the left, followed by a home icon and the text 'nicholasrenotte — -bash — 80x20'. The terminal content shows a green prompt 'Last login: Tue Jan 22 17:27:01 on ttys001' followed by a green command prompt 'Nicholass-MacBook-Pro:~ nicholasrenotte\$'. The command 'pip install watson-developer-cloud' has been entered and is highlighted in green. The rest of the terminal area is dark and empty.

```
nicholasrenotte — -bash — 80x20
Last login: Tue Jan 22 17:27:01 on ttys001
Nicholass-MacBook-Pro:~ nicholasrenotte$ pip install watson-developer-cloud
```

STEP 5 – CLASSIFYING GENERAL IMAGES

It's now time to start classifying some stuff. This example uses jupyter notebooks to interact with the API. If you're not familiar with how to use notebooks, check out this [quick tutorial](#).

The first step is to import the `json` module and the visual recognition method from the `watson_developer_cloud` module.

```
import json
```

```
from watson_developer_cloud import VisualRecognitionV3
```

Then create a new instance of the visual recognition service and update *iam_apikey* to the apikey you generated in Step 3.

```
# Create instance of VR Service
```

```
visual_recognition = VisualRecognitionV3(
```

```
'2018-03-19',
```

```
iam_apikey='gdFV6R4ssFNjyZ6eGzBompYQ8DyFC9OfwAFDr4P3qppF') # Replace this with your APIKEY
```

Grab a url of an image you'd like to classify and update the url variable. Then run the classify method against the visual_recognition service to retrieve the classification.

```
# URL that you want to classify
```

```
url = 'https://cdn.shopclues.com/images/thumbnails/18729/320/320/78279966PC14275391591432900471.jpg'
```

```
# Call classify method from service
```

```
classes_result = visual_recognition.classify(url=url).get_result()
```

```
# Pretty print JSON result
```

```
print(json.dumps(classes_result, indent=2))
```

The response can be pretty-printed using `json.dumps` and should look similar to the result shown below. The image used was a basic desktop computer that looked something like this:



Looking at the classes returned you can see that the classifier accurately classified the image as a desktop computer.

```
# Check classes returned
```

```
classes_result['images'][0]['classifiers'][0]['classes']
```

```
# Expected results
```

```
[{'class': 'desktop computer',
```

```
  'score': 0.959,
```

```
  'type_hierarchy': '/machine/computer/digital computer/personal computer/desktop computer'},
```

```
 {'class': 'personal computer', 'score': 0.977},
```

```
 {'class': 'digital computer', 'score': 0.977},
```

```
 {'class': 'computer', 'score': 0.984},
```

```
 {'class': 'machine', 'score': 0.984},
```

```
 {'class': 'system', 'score': 0.77},
```

```
 {'class': 'coal black color', 'score': 0.901}]
```

STEP 6 – CLASSIFYING FOOD

One of the cool things about the visual recognition API is that you're able to use more than just their basic classifier. For example the devs over at IBM have also developed a classifier that allows you to classify food.

To use that classifier just pass through an extra argument to the classify method. The argument required is *classifier_ids=["food"]*.

```
# Food URL that you want to classify
```

```
l.com/wp-content/uploads/2017/10/orange.jpg'
```

```
# Call classify method from service with classifier_ids parameter set
```

```
classes_result = visual_recognition.classify(url=url, classifier_ids=["food"]).get_result()
```

```
# Pretty print JSON result
```

```
p
```

When you run this classifier you'll actually notice that the response shows that the classifier_id being used is the food classifier. This might not seem all that important now but it becomes increasingly important when you start training your own models.

```
{
  "images": [
    {
      "classifiers": [
        {
          "classifier_id": "food",
          "name": "food",
          "classes": [
            {
              "class": "orange",
              "score": 0.799,
              "type_hierarchy": "/fruit/citrus/orange"
            },
            {
              "class": "citrus"
            }
          ]
        }
      ]
    }
  ]
}
```

STEP 7 – DETECTING FACES

Watson VR also opens up the ability to detect faces. Rather than using the classify method we'll use detect_faces to leverage the facial recognition api. One of the really neat things about this function is that it actually picks up multiple faces within a particular image and also estimates the gender and the age of that face!

To switch our code over so that it can detect faces, simply change the classify method to detect_faces and run the code. (NB: Update the url to one that has images of faces as well)

```
# Face URL that you want to classify
```

```
url = 'https://upload.wikimedia.org/wikipedia/commons/thumb/2/2a/Donald\_Glover\_TIFF\_2015.jpg/220px-Donald\_Glover\_TIFF\_2015.jpg'
```

```
# Call detect faces method from service
```

```
classes_result = visual_recognition.detect_faces(url=url).get_result()
```

```
# Pretty print JSON result
```

```
print(json.dumps(classes_result, indent=2))
```

The response should return an array of faces as well as the estimated age of that person, the location of the person's face within the photo and the guestimated age

```
{
  "images": [
    {
      "faces": [
        {
          "age": {
            "min": 26,
            "max": 29,
            "score": 0.8627813
          },
          "face_location": {
            "height": 125,
            "width": 116,
            "left": 56,
            "top": 80
          },
          "gender": {
            "gender": "MALE",
            "gender_label": "male",
            "score": 0.9999863
          }
        }
      ]
    }
  ],
  1,
}
```

STEP 8 – CUSTOM CLASSIFICATIONS

“This is great Nick, but i don’t really care about classifying fruit or detecting faces”

Well, you’re in luck because the visual recognition API allows you to train custom classes. This would be like [training your own CNN using Tensorflow](#) but way easier.

To do this, you need to load zip files containing images of what you’re trying to classify as well as images of things that don’t form part of that class. Then run the `create_classifier` method against your visual recognition service.

```
# Open each image zip file
```

```
with open('./beagle.zip', 'rb') as beagle, \
```

```
open('./golden-retriever.zip', 'rb') as goldenretriever, \
```

```
open('./husky.zip', 'rb') as husky, \
```

```
open('./cats.zip', 'rb') as cats:
```

```
# Create new classifier category
```

```
model = visual_recognition.create_classifier('dogs',
```

```
beagle_positive_examples=beagle,  
  
goldenretriever_positive_examples=goldenretriever,  
  
husky_positive_examples=husky,  
  
negative_examples=cats).get_result()
```

```
# Pretty print JSON result
```

```
print(json.dumps(model, indent=2))
```

Assuming everything went well this will return a response that shows that the model has started training.

```
{  
  "classifier_id": "dogs_33552121",  
  "name": "dogs",  
  "status": "training",  
  "owner": "98bf5182-2cd7-45a7-8935-3000f238ffe2",  
  "created": "2019-01-23T06:38:58.616Z",  
  "updated": "2019-01-23T06:38:58.616Z",  
  "classes": [  
    {  
      "class": "husky"  
    },  
    {  
      "class": "goldenretriever"  
    },  
    {  
      "class": "beagle"  
    }  
  ],  
  "core_ml_enabled": true  
}
```

We can check on the training status using the `get_classifier` method.

```
# Check status of classifier
```

```
classifier = visual_recognition.get_classifier(classifier_id='dogs_33552121').get_result()
```

```
# Pretty print JSON result
```

```
print(json.dumps(classifier, indent=2))
```



```
{
  "classifier_id": "dogs_33552121",
  "name": "dogs",
  "status": "ready",
  "owner": "98bf5182-2cd7-45a7-8935-3000f238ffe2",
  "created": "2019-01-23T06:38:58.616Z",
  "updated": "2019-01-23T06:38:58.616Z",
  "classes": [
    {
      "class": "husky"
    },
    {
      "class": "goldenretriever"
    },
    {
      "class": "beagle"
    }
  ],
  "core_ml_enabled": true
}
```

Once the status has changed from training to ready we can use the model to classify custom images in this case, pictures of dogs.

```
# URL from custom class that you want to classify
```

```
url = 'https://i.ytimg.com/vi/bx7BjjqHf2U/maxresdefault.jpg'
```

```
# Run classifier using classifier ID from custom classifier result
```

```
result = visual_recognition.classify(url=url, classifier_ids=["dogs_33552121"]).get_result()
```

```
# Pretty print JSON result
```

```
print(json.dumps(result, indent=2))
```

```
{
  "images": [
    {
      "classifiers": [
        {
          "classifier_id": "dogs_33552121",
          "name": "dogs",
          "classes": [
            {
              "class": "beagle",
              "score": 0.905
            }
          ]
        }
      ],
      "source_url": "https://i.ytimg.com/vi/bx7BjjqHf2U/maxresdefault.jpg",
      "resolved_url": "https://i.ytimg.com/vi/bx7BjjqHf2U/maxresdefault.jpg"
    }
  ],
  "images_processed": 1,
  "custom_classes": 3
}
```

To build an image recognition system using IBM Cloud Visual Recognition and AI-generated captions

This are Following Steps:

1. Collect and reprocess your image dataset.
2. Train your Visual Recognition model (if necessary).
3. Integrate the IBM Visual Recognition API into your application.
4. Perform image classification using the API.
5. Use a natural language generation model to create captions for the recognized images.
6. Display or store the generated captions alongside the images.

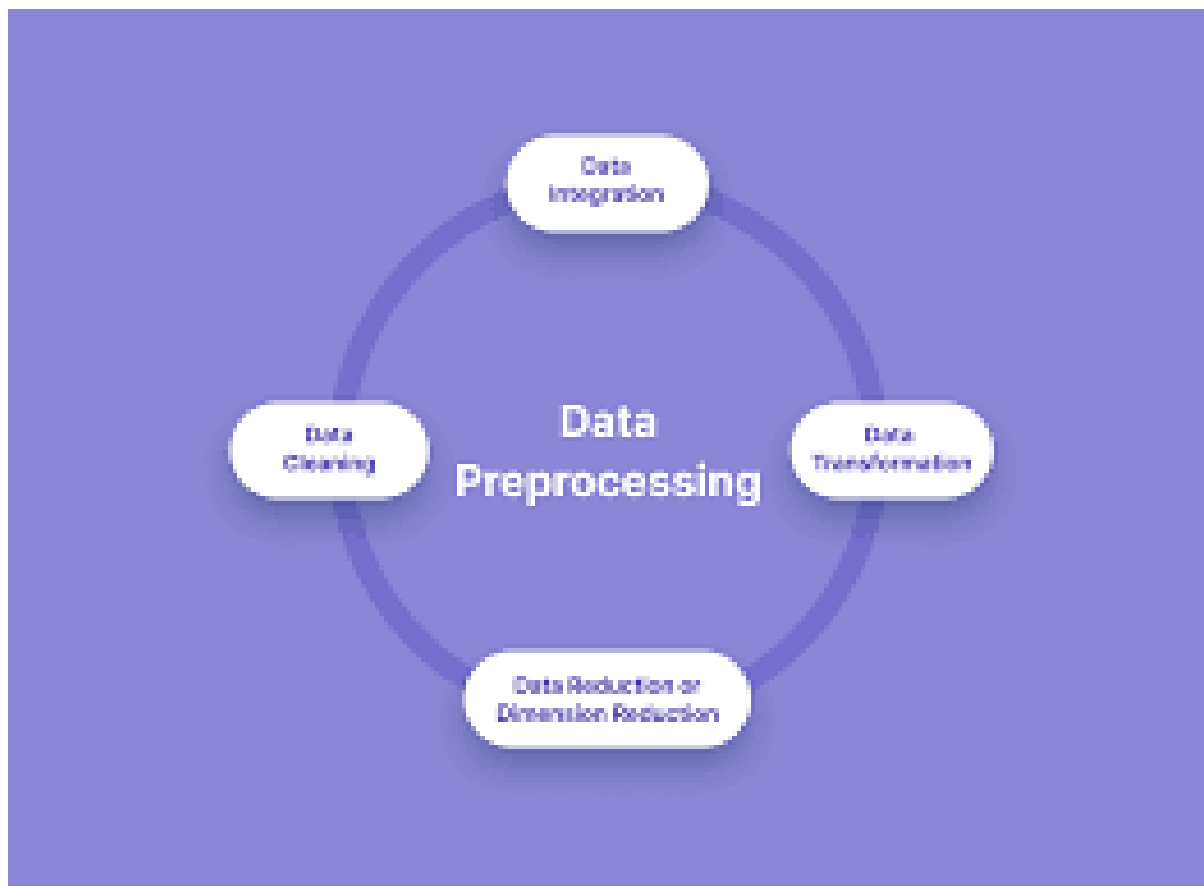
Collect and reprocess your image dataset:-

Collecting Images:



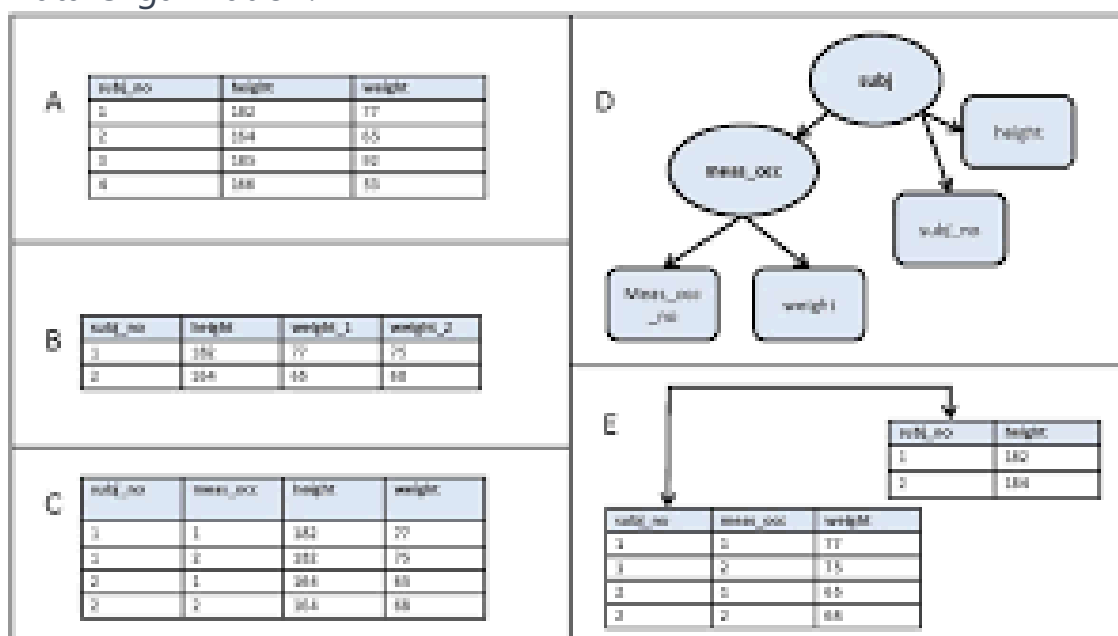
Acquire a set of images that you want to use for classification and captioning. These images should be relevant to your application or use case.

Preprocessing Images:



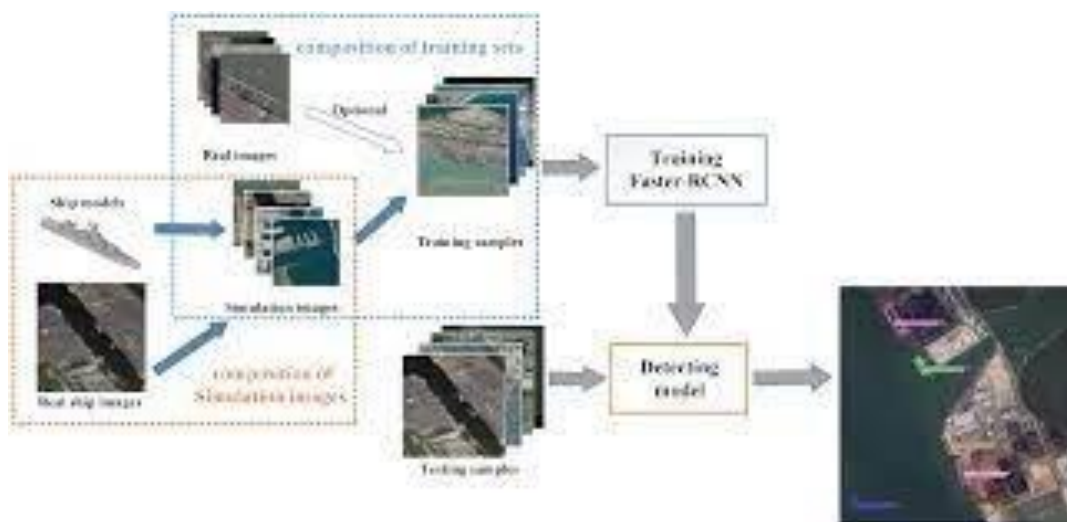
Before using these images, it's important to ensure they are well-preprocessed. This typically includes tasks such as resizing images to a consistent format, removing noise, and standardizing file formats.

Data Organization:



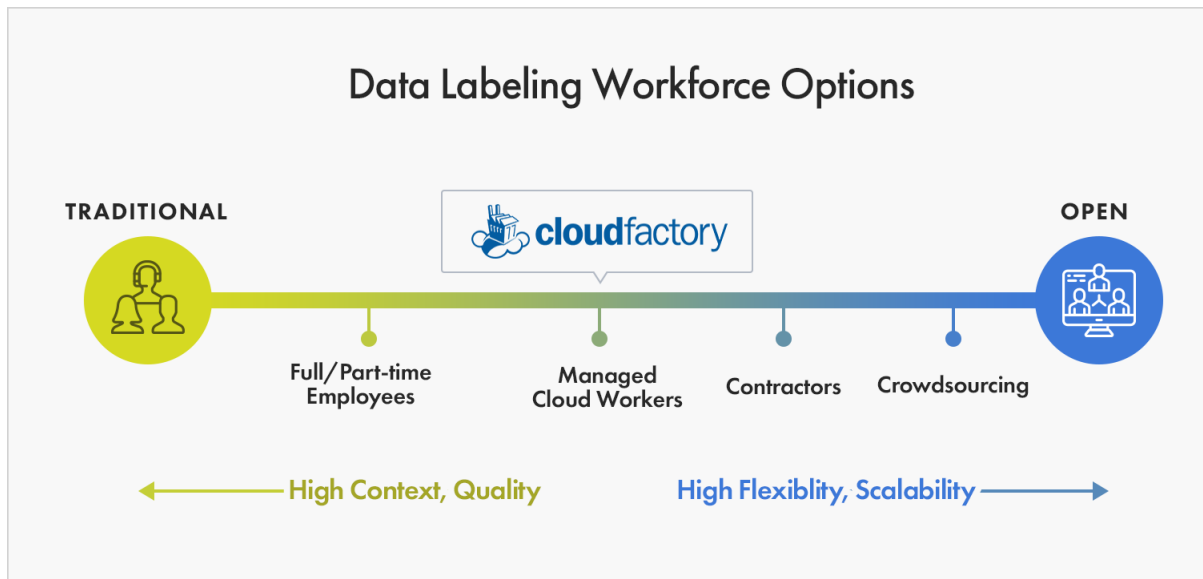
Organize your images in a structured manner, possibly into folders or directories that correspond to different categories or classes. This structure will make it easier to train and test your image recognition model.

Data Augmentation (Optional):



Depending on the quality and quantity of your dataset, you might consider data augmentation techniques, like rotating, flipping, or cropping images to create variations. This can improve the model's robustness.

Data Labeling:



Assign labels or categories to each image in your dataset. In the context of IBM Cloud Visual Recognition, these labels will be used for classification. Ensure that the labeling is accurate and matches the content of each image.

Quality Control:

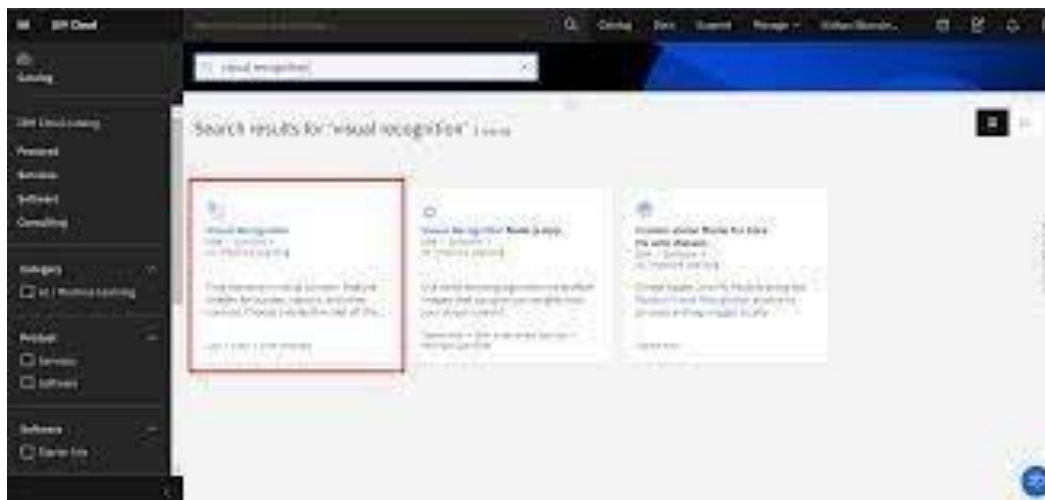


Review your dataset to identify and eliminate any outliers, inaccuracies, or irrelevant images that may negatively impact the model's performance.

Once you've completed these steps, your image dataset will be well-prepared for training and testing your image recognition model with IBM Cloud Visual Recognition.

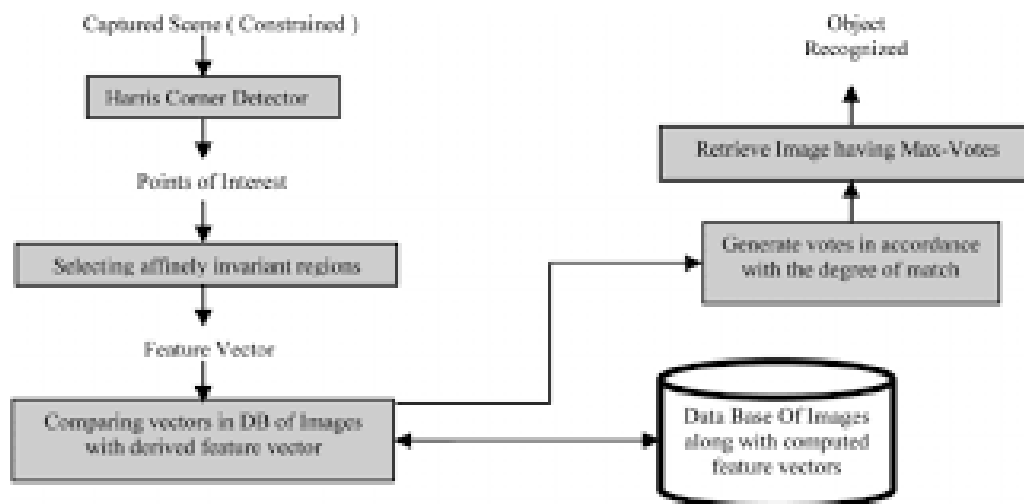
Use natural language generation to create captions for the recognized images.

Image Recognition with IBM Visual Recognition:



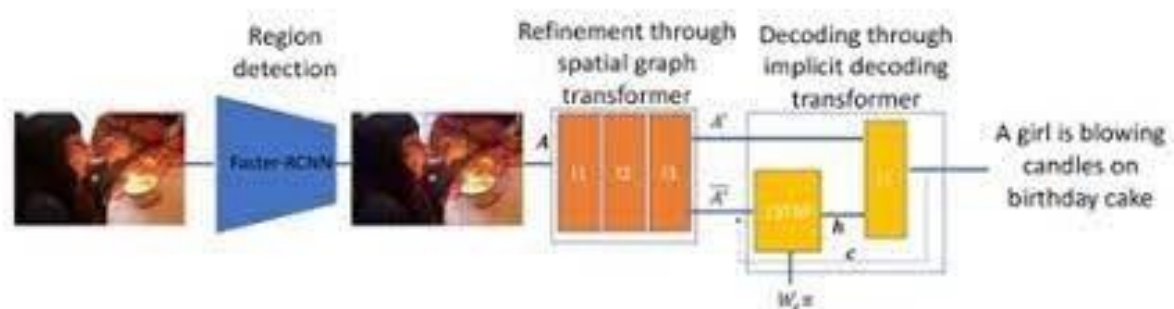
Use the IBM Cloud Visual Recognition API to analyze and recognize the content of an image. This step involves sending an image to the API and receiving a list of recognized objects or classes along with confidence scores.

Selecting Recognized Objects:



Identify the objects or classes that you want to include in the image caption. You can select the top-ranked objects based on their confidence scores or choose specific objects that are most relevant to the image.

Generate Image Caption with NLG:



Utilize an NLG model or service to generate a natural language caption based on the recognized objects. OpenAI's GPT-3 is an example of an NLG model that can be used for this purpose.

Here's an example of how you can generate a caption with GPT-3 in Python:

```
python
```

```
Copy code
```

```
import openai
```



```
openai.api_key = 'YOUR_OPENAI_API_KEY'
```

```
def generate_caption(objects):
```

```
    prompt = f"Describe an image with {' '.join(objects)} in it."
```

```
    response = openai.Completion.create(
```

```
        engine="davinci",
```

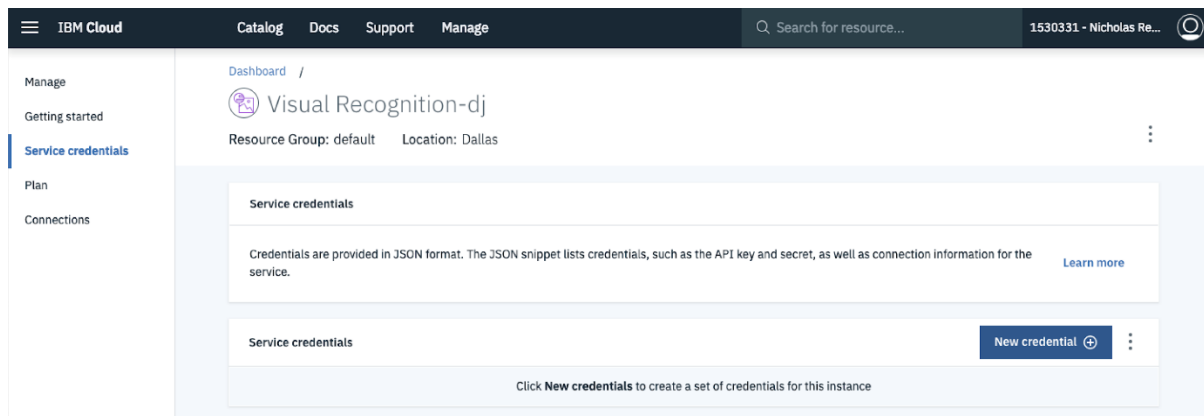
```
        prompt=prompt,
```

```
        max_tokens=50 # Adjust the desired caption length
```

```
    )
```

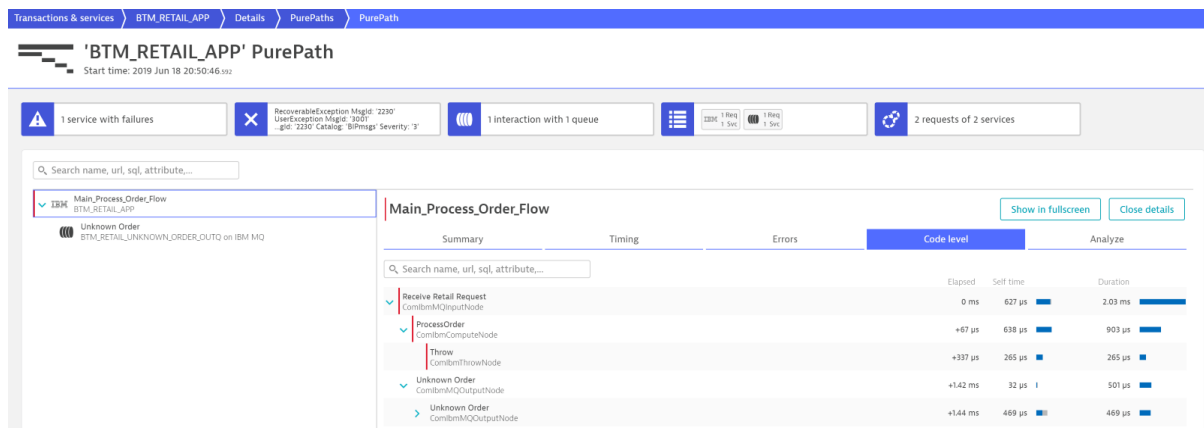
```
    return response.choices[0].text
```

Formatting and Display:



Format the generated caption as needed and display it alongside the recognized image. You can combine the objects recognized by IBM Visual Recognition with the NLG-generated caption for a complete and descriptive result.

Error Handling and Fine-Tuning:



Be prepared to handle cases where certain objects are not recognized or where the caption generation may not produce the desired quality. You can fine-tune your NLG model or add fallback mechanisms to improve the overall system's performance.

By integrating image recognition with NLG, you can automatically create meaningful captions for recognized images, enhancing their description and usability in various applications such as content generation, accessibility, and more.