Colorizing Black & White Images Using GAN

From Team Apes of AI

Anurag Kesavaraju

Mahitha Rudraraju

Naveen Somavarapu

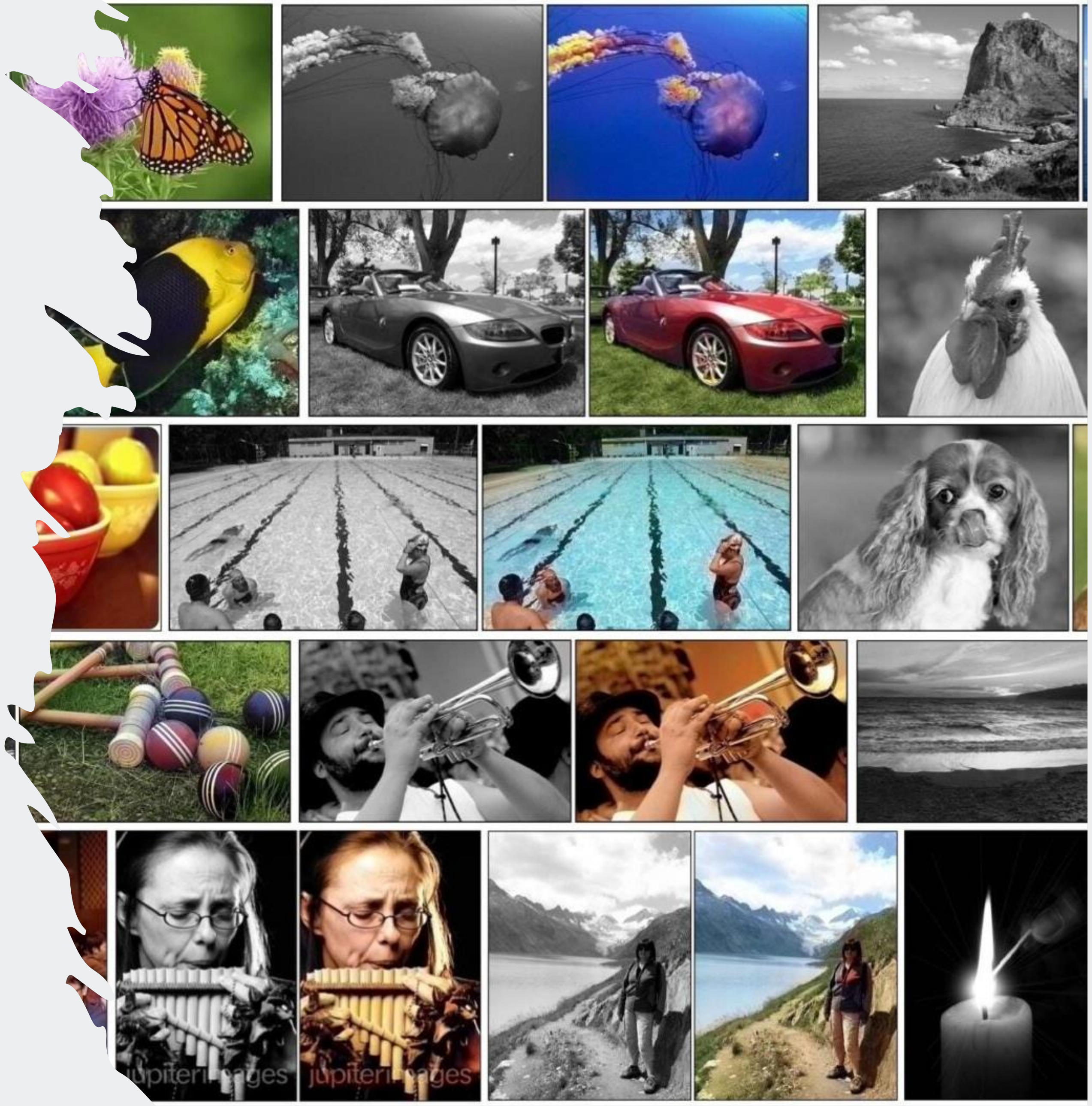
Sai Chaitanya Nagandla

Sai Dheeraja Rangineni

Georgia State University | J. Mack Robinson College of Business

Problem Statement

- Generate visually appealing images by converting Black and White images to Color.
- Cultural Heritage and Museums can have historic photographs to be more visually appealing.
- Users can enjoy old Black & White movies in color.



Introduction to colorization problem

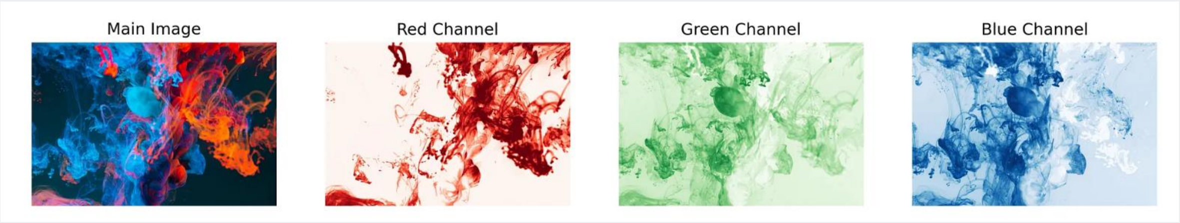
- In L*a*b color space, we have three numbers for each pixel, but these numbers have different meanings.

  - L : Lightness of each pixel (the second image in the row)
  - *a and *b channels: Encodes how much green-red and yellow-blue each pixel is

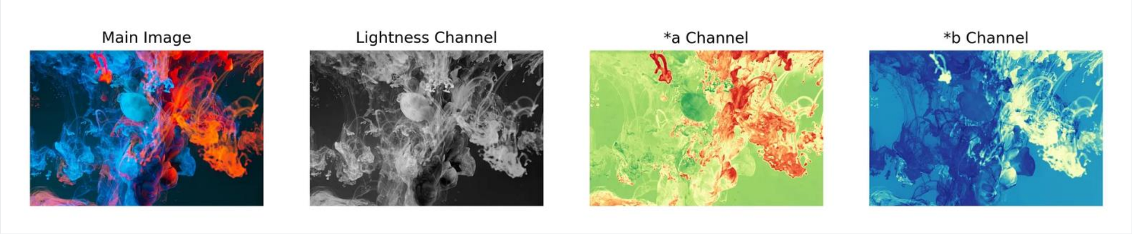
- When using L*a*b, L channel becomes input to the model and two channels (*a, *b) will be predicted.

- Later, we concatenate all the channels, and we get our colorful image.

- If you use RGB, you must first convert your image to grayscale, feed the grayscale image to the model and hope it will predict 3 numbers.

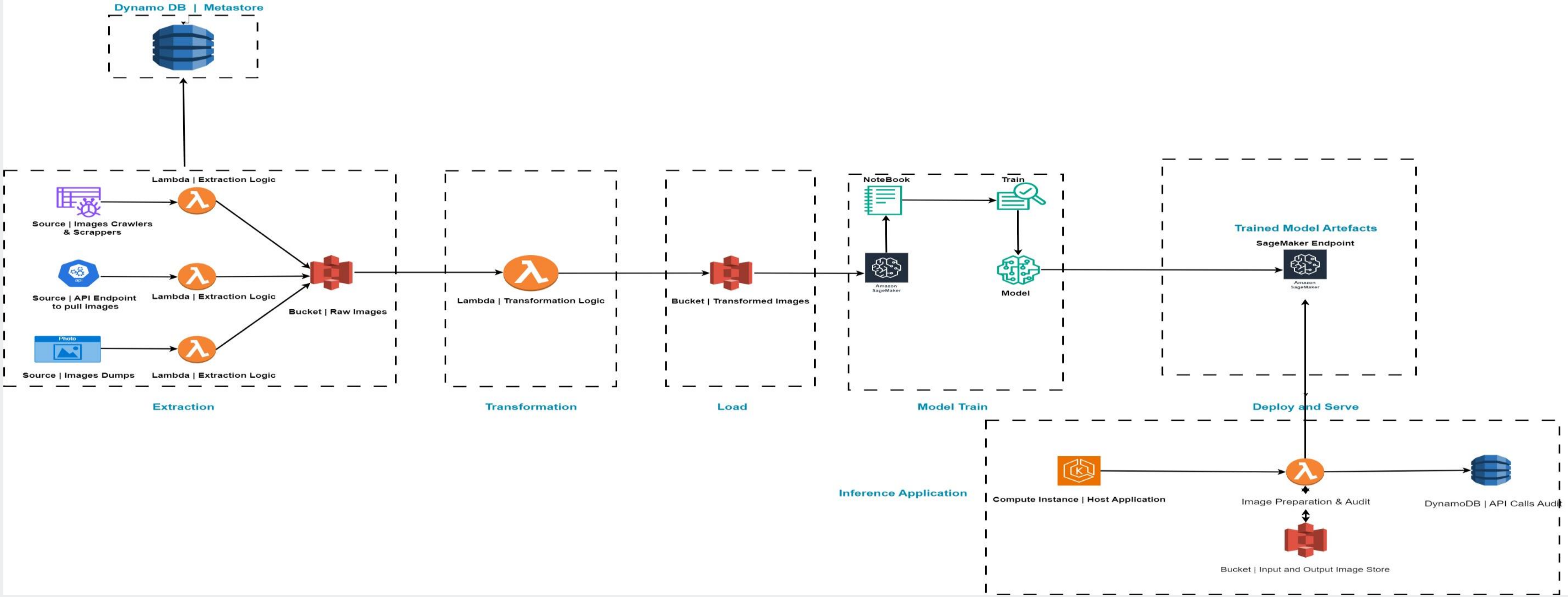


RGB



L*a*b

Architecture & Flow



Data Source

Source A: Imagenet

ImageNet is a vast image database with over 14 million annotated pictures, pivotal for training and benchmarking computer vision models. It organizes images according to the WordNet hierarchy, crucial for advancing object recognition and deep learning research.

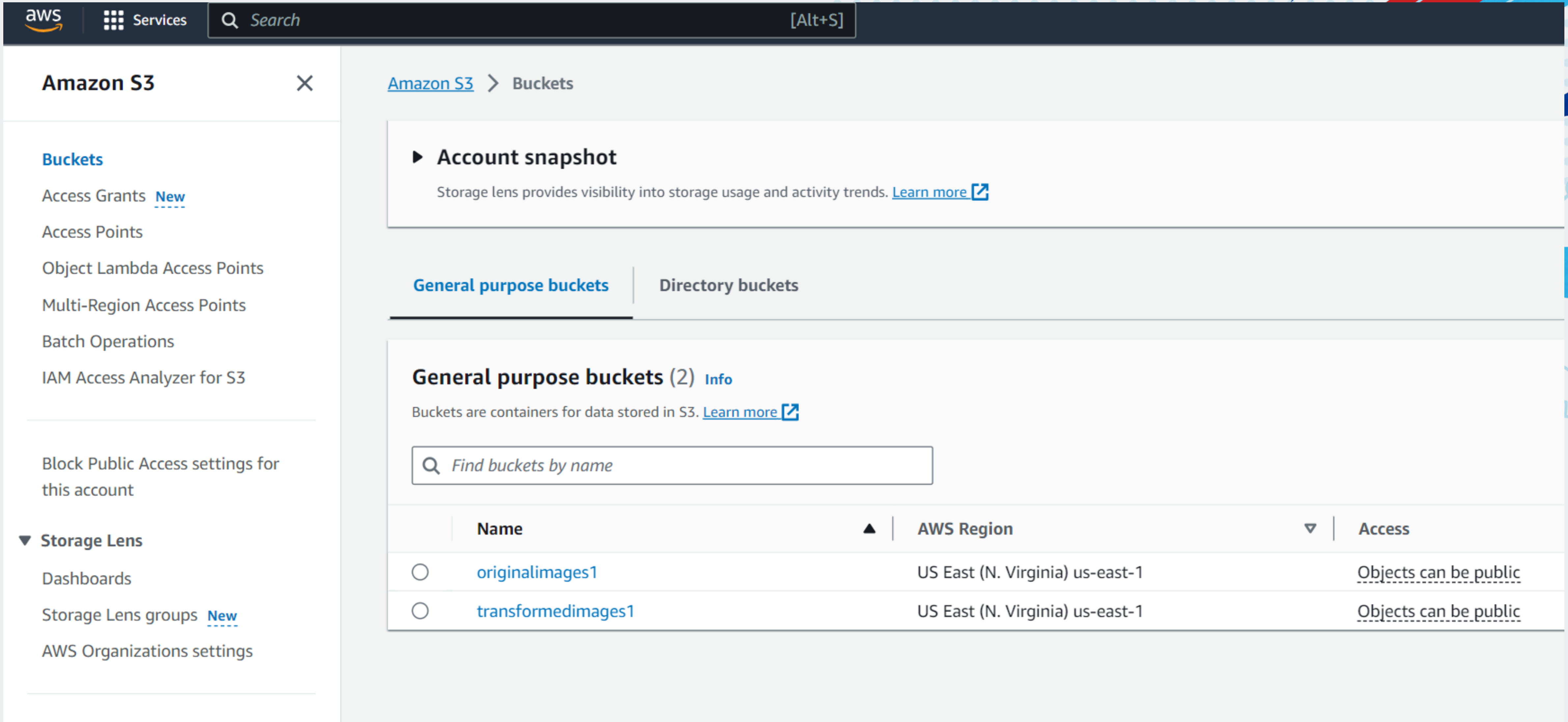
Source B: COCO

COCO (Common Objects in Context) dataset: Over 200,000 annotated images for object detection, segmentation, and captioning, vital for training and testing computer vision algorithms in real-world scenarios

Source C: Flickr

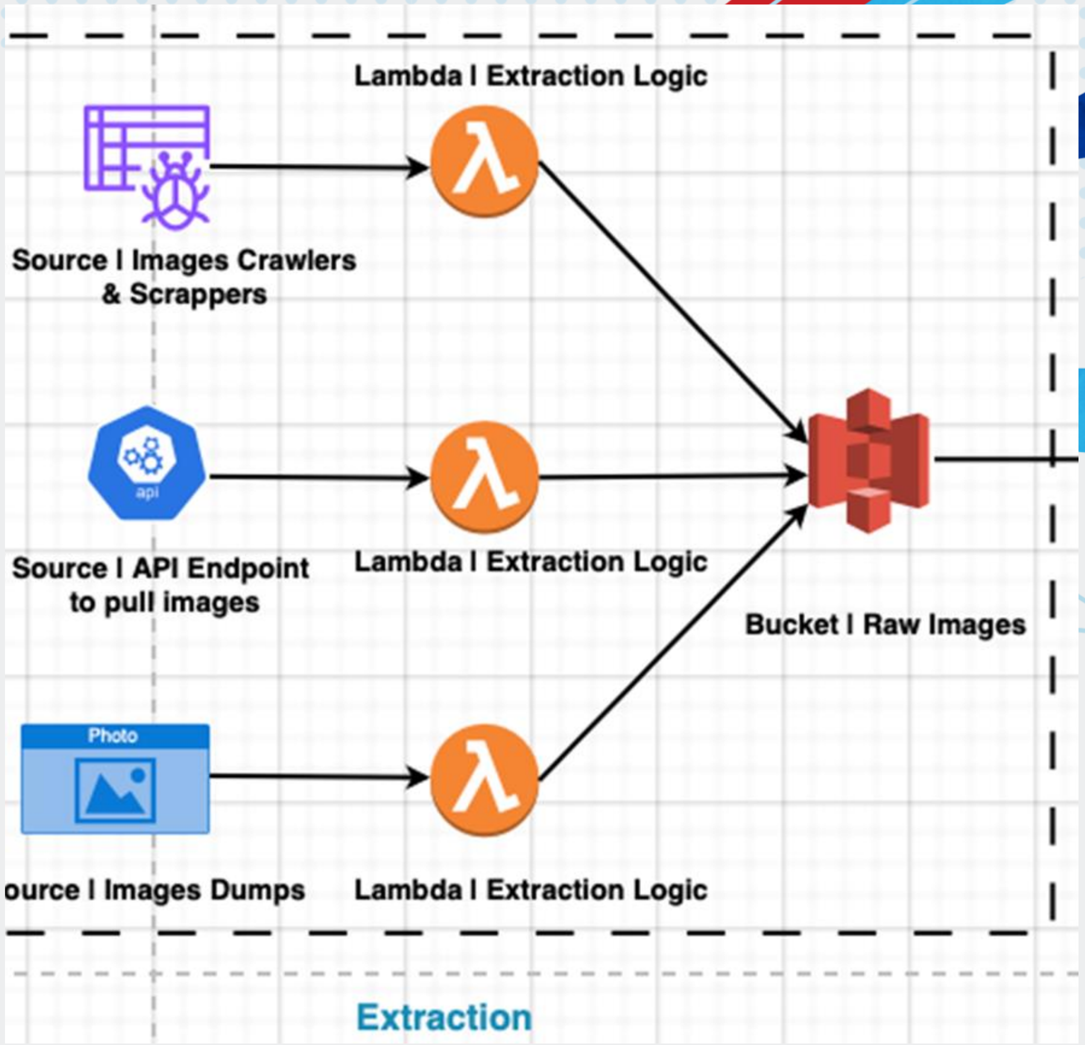
Online platform for sharing photos and videos, offering a vast collection of user-generated visual content." "It's utilized for research, creativity, and personal sharing, serving as a rich resource for diverse purposes.

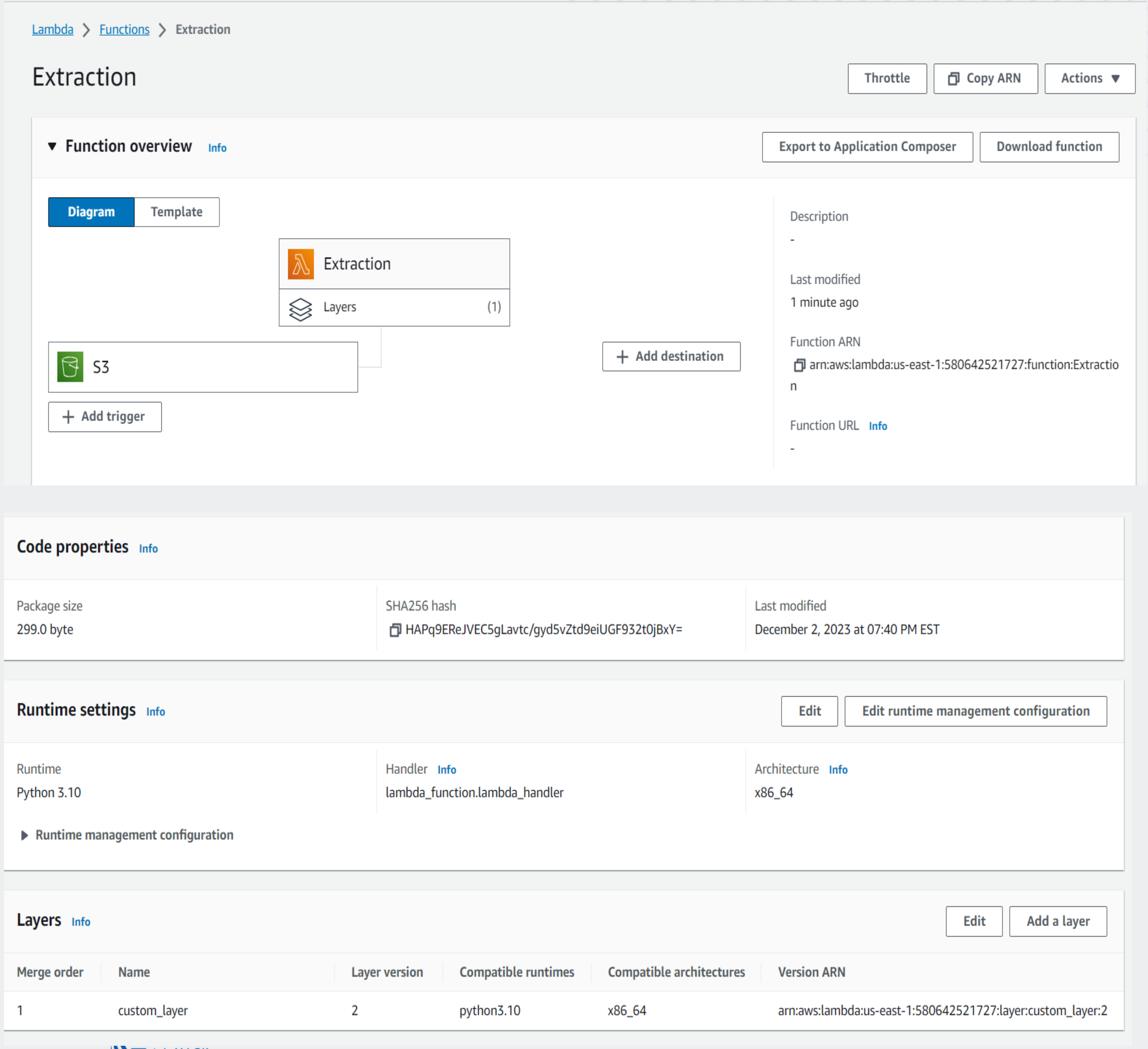
Data Storage (S3)



Extraction

- Diverse data sources were used for this use-case.
- Below are the main sources
    1. COCO Dataset : Image dumps
    2. ImageNET
    3. API endpoints: Flickr





- Each data source has a dedicated Lambda function to extract the data.
- Update the required details such as the data source, S3 bucket details.. etc.
- Now, the Lambda function is ready to use.
- A layer has been created to call additional libraries such as pillow, OpenCV..etc.

Transformation and Load

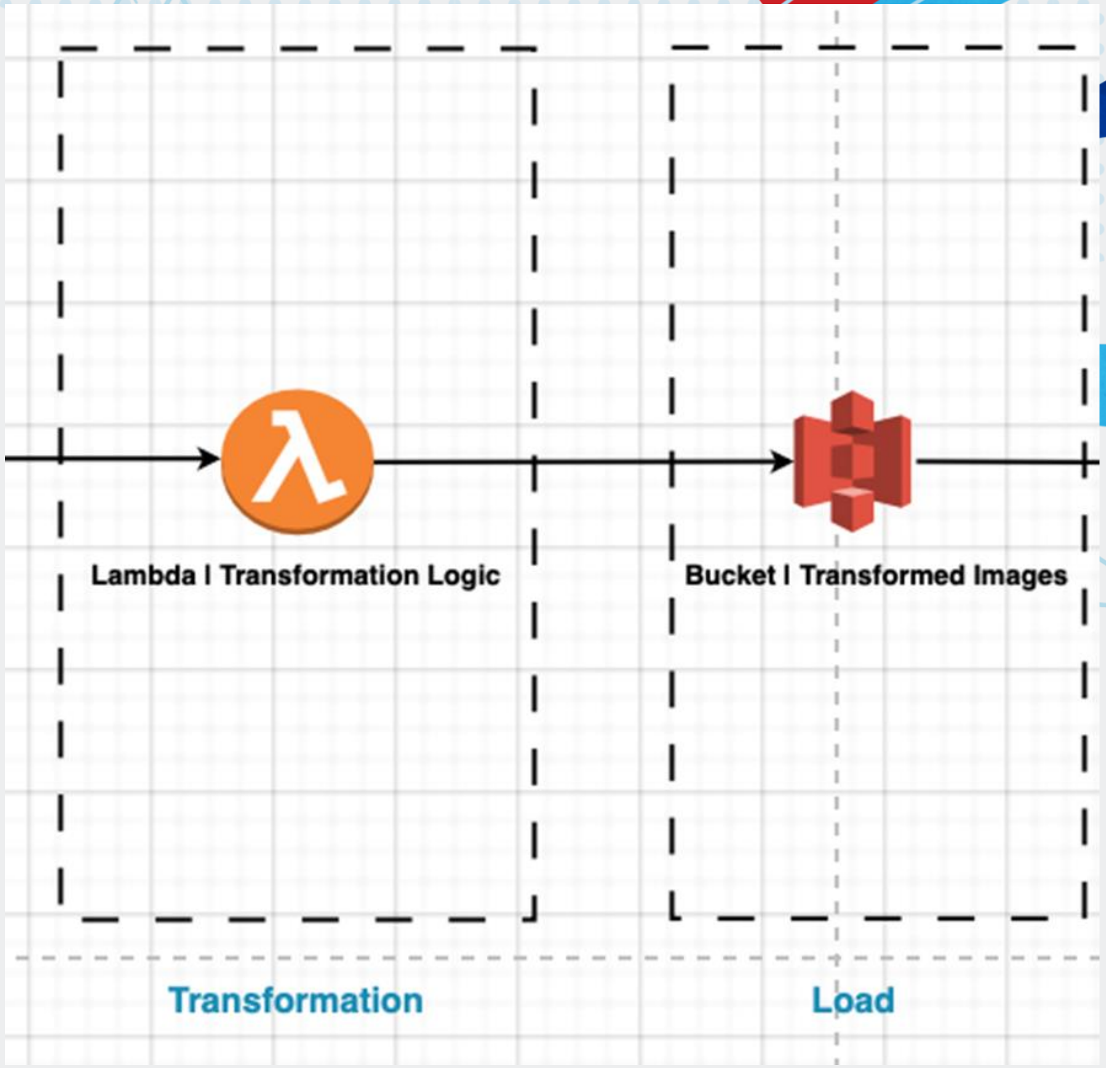
- Transformation logic involves the following tasks on

raw data

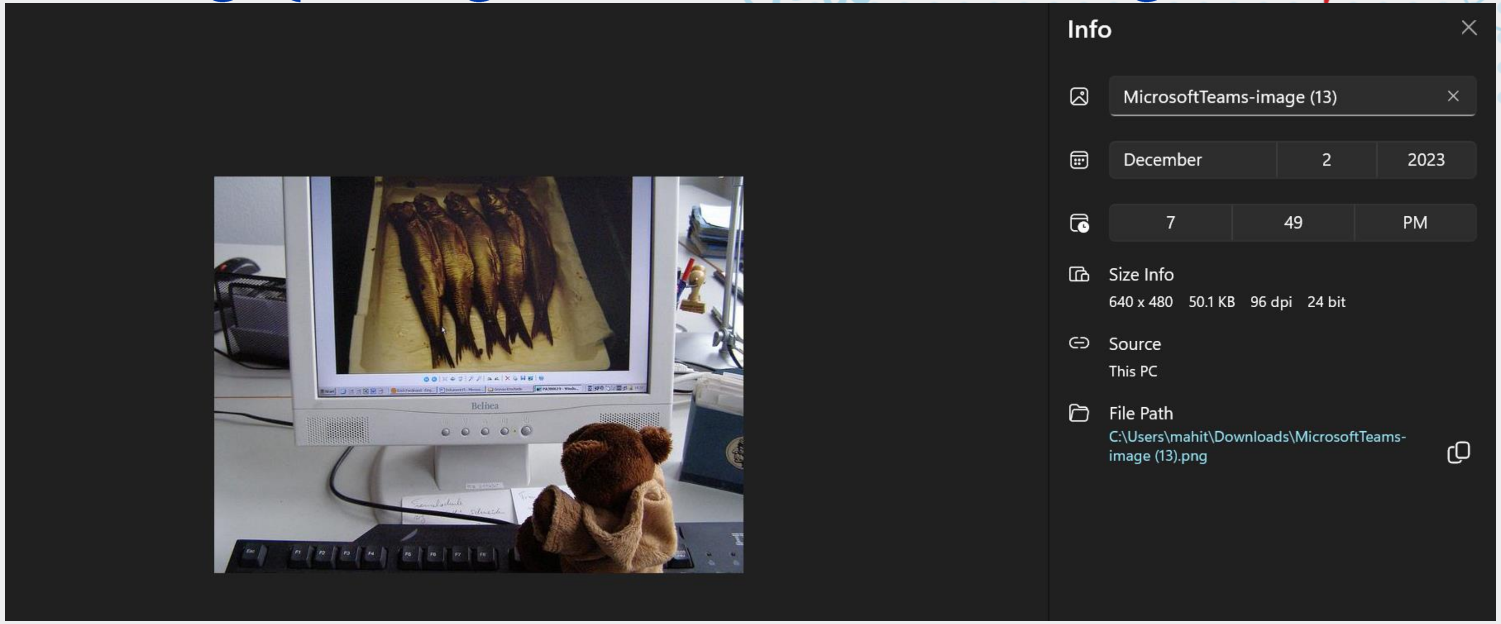
1. Resizing the image to 256*256 pixels
2. Random horizontal flipping
3. Converting images from RGB color space to L*a*b
4. Scaling pixel values to the range [-1, 1]

- Python script once done with transformations, loads

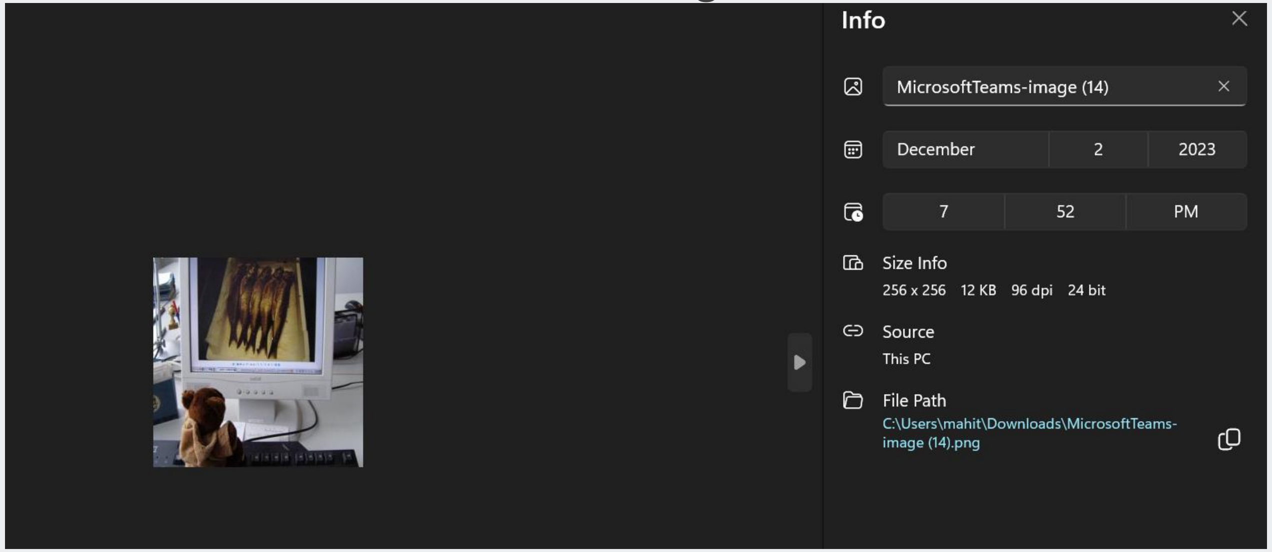
The data to another S3 bucket using Lambda function



Pre-processing (Image Resize and Augmentation)



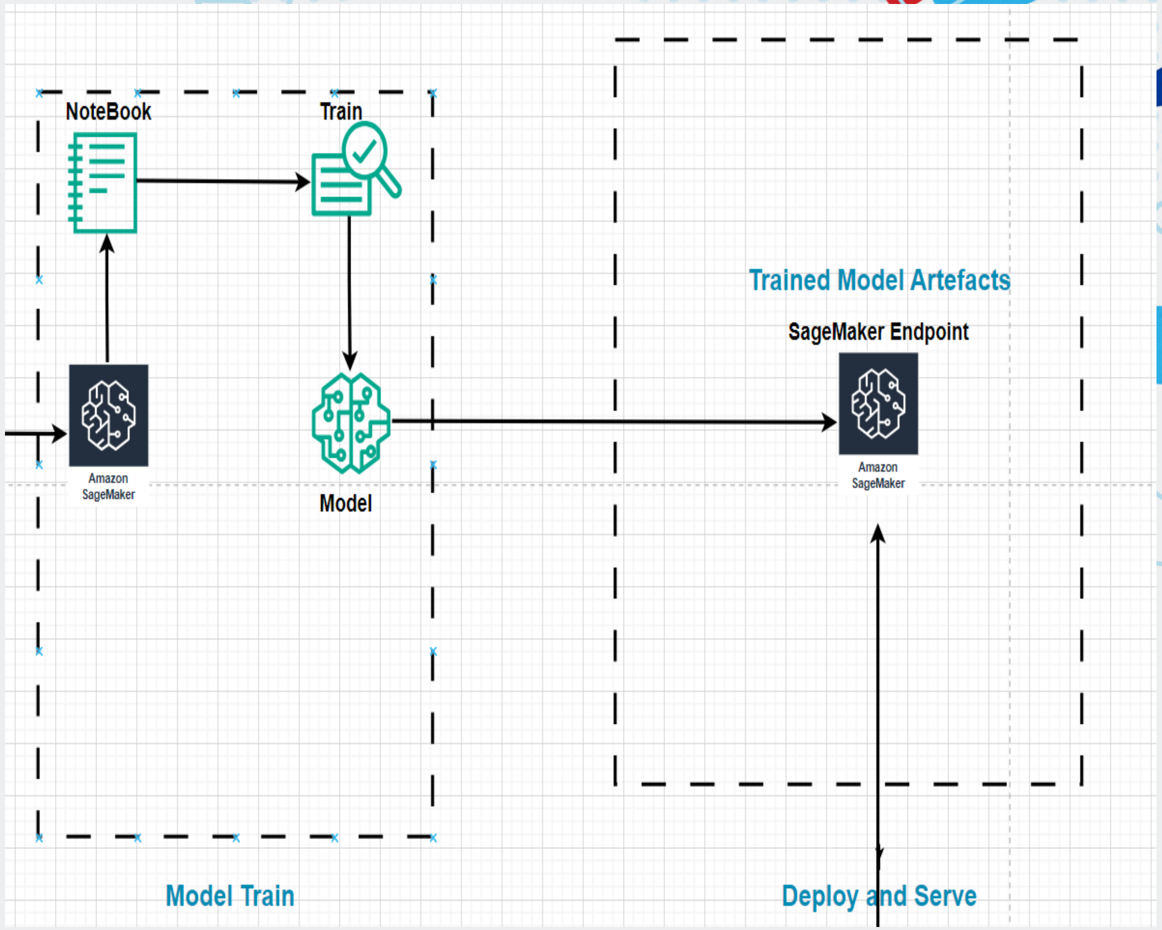
Actual Image



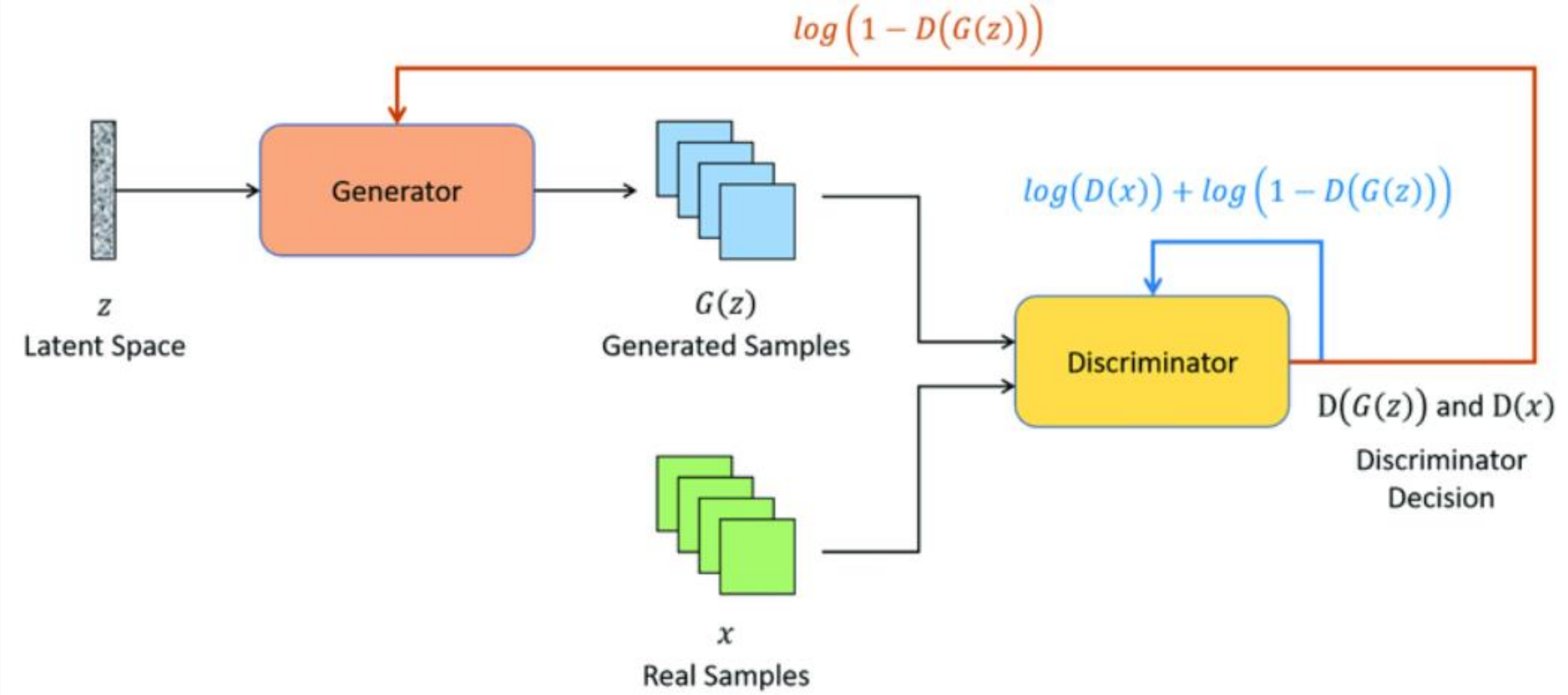
Resized Image

Model Train and Deploy

- Initially for prototyping, the model was trained within Colab Notebook.
- After the prototyping, they were incorporated and then trained on AWS Sagemaker.
- GAN losses and metrics were reviewed and the model was published to an Endpoint to be served within Sagemaker.



GAN



$log\left(1 - D(G(z))\right)$
Generator
$log(D(x)) + log\left(1 - D(G(z))\right)$
z
Latent Space
$G(z)$
Generated Samples
Discriminator
$D(G(z))$ and $D(x)$
Discriminator Decision
x
Real Samples

GAN Generator

GAN Down Sampler

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 128, 128]	1,024
LeakyReLU-2	[-1, 64, 128, 128]	0
Conv2d-3	[-1, 128, 64, 64]	131,072
BatchNorm2d-4	[-1, 128, 64, 64]	256
LeakyReLU-5	[-1, 128, 64, 64]	0
Conv2d-6	[-1, 256, 32, 32]	524,288
BatchNorm2d-7	[-1, 256, 32, 32]	512
LeakyReLU-8	[-1, 256, 32, 32]	0
Conv2d-9	[-1, 512, 16, 16]	2,097,152
BatchNorm2d-10	[-1, 512, 16, 16]	1,024
LeakyReLU-11	[-1, 512, 16, 16]	0
Conv2d-12	[-1, 512, 8, 8]	4,194,304
BatchNorm2d-13	[-1, 512, 8, 8]	1,024
LeakyReLU-14	[-1, 512, 8, 8]	0
Conv2d-15	[-1, 512, 4, 4]	4,194,304
BatchNorm2d-16	[-1, 512, 4, 4]	1,024
LeakyReLU-17	[-1, 512, 4, 4]	0
Conv2d-18	[-1, 512, 2, 2]	4,194,304
BatchNorm2d-19	[-1, 512, 2, 2]	1,024
LeakyReLU-20	[-1, 512, 2, 2]	0
Conv2d-21	[-1, 512, 1, 1]	4,194,304
ReLU-22	[-1, 512, 1, 1]	0

GAN Down Up-sampler

Conv2d-21	[-1, 512, 1, 1]	4,194,30
ReLU-22	[-1, 512, 1, 1]	
ConvTranspose2d-23	[-1, 512, 2, 2]	4,194,30
BatchNorm2d-24	[-1, 512, 2, 2]	1,02
UnetBlock-25	[-1, 1024, 2, 2]	
ReLU-26	[-1, 1024, 2, 2]	
ConvTranspose2d-27	[-1, 512, 4, 4]	8,388,60
BatchNorm2d-28	[-1, 512, 4, 4]	1,02
Dropout-29	[-1, 512, 4, 4]	
UnetBlock-30	[-1, 1024, 4, 4]	
ReLU-31	[-1, 1024, 4, 4]	
ConvTranspose2d-32	[-1, 512, 8, 8]	8,388,60
BatchNorm2d-33	[-1, 512, 8, 8]	1,02
Dropout-34	[-1, 512, 8, 8]	
UnetBlock-35	[-1, 1024, 8, 8]	
ReLU-36	[-1, 1024, 8, 8]	
ConvTranspose2d-37	[-1, 512, 16, 16]	8,388,60
BatchNorm2d-38	[-1, 512, 16, 16]	1,02
Dropout-39	[-1, 512, 16, 16]	
UnetBlock-40	[-1, 1024, 16, 16]	
ReLU-41	[-1, 1024, 16, 16]	
ConvTranspose2d-42	[-1, 256, 32, 32]	4,194,30
BatchNorm2d-43	[-1, 256, 32, 32]	51
UnetBlock-44	[-1, 512, 32, 32]	
ReLU-45	[-1, 512, 32, 32]	
ConvTranspose2d-46	[-1, 128, 64, 64]	1,048,57
BatchNorm2d-47	[-1, 128, 64, 64]	25
UnetBlock-48	[-1, 256, 64, 64]	
ReLU-49	[-1, 256, 64, 64]	
ConvTranspose2d-50	[-1, 64, 128, 128]	262,14
BatchNorm2d-51	[-1, 64, 128, 128]	12
UnetBlock-52	[-1, 128, 128, 128]	
ReLU-53	[-1, 128, 128, 128]	

GAN Discriminator

```
  (model): Sequential(
    (0): Sequential(
      (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (2): Sequential(
      (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (3): Sequential(
      (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(1, 1), padding=(1, 1), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2, inplace=True)
    )
    (4): Sequential(
      (0): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), padding=(1, 1))
    )
  )
)
```

GAN Loss

```
print(f'Epoch {epoch + 1}, GAN Loss: {loss_value.item()}')
```

```
GAN Loss: 1.0033090114593506
Epoch 1, GAN Loss: 1.0033090114593506
GAN Loss: 1.0004392862319946
Epoch 2, GAN Loss: 1.0004392862319946
GAN Loss: 0.9964740872383118
Epoch 3, GAN Loss: 0.9964740872383118
GAN Loss: 0.9372398853302002
Epoch 4, GAN Loss: 0.9372398853302002
GAN Loss: 1.0335652828216553
Epoch 5, GAN Loss: 1.0335652828216553
GAN Loss: 0.9920356869697571
Epoch 6, GAN Loss: 0.9920356869697571
GAN Loss: 1.020262360572815
Epoch 7, GAN Loss: 1.020262360572815
GAN Loss: 0.9837158918380737
Epoch 8, GAN Loss: 0.9837158918380737
GAN Loss: 0.9967315196990967
Epoch 9, GAN Loss: 0.9967315196990967
GAN Loss: 1.025168538093567
Epoch 10, GAN Loss: 1.025168538093567
```

- The GAN loss values generally seem to decrease from epoch to epoch. This is a positive sign, suggesting that the generator is improving over time in generating samples that are more convincing to the discriminator.

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z)))]$$

- While the values generally decrease, there are some fluctuations. It's normal to see some variability in GAN loss during training.

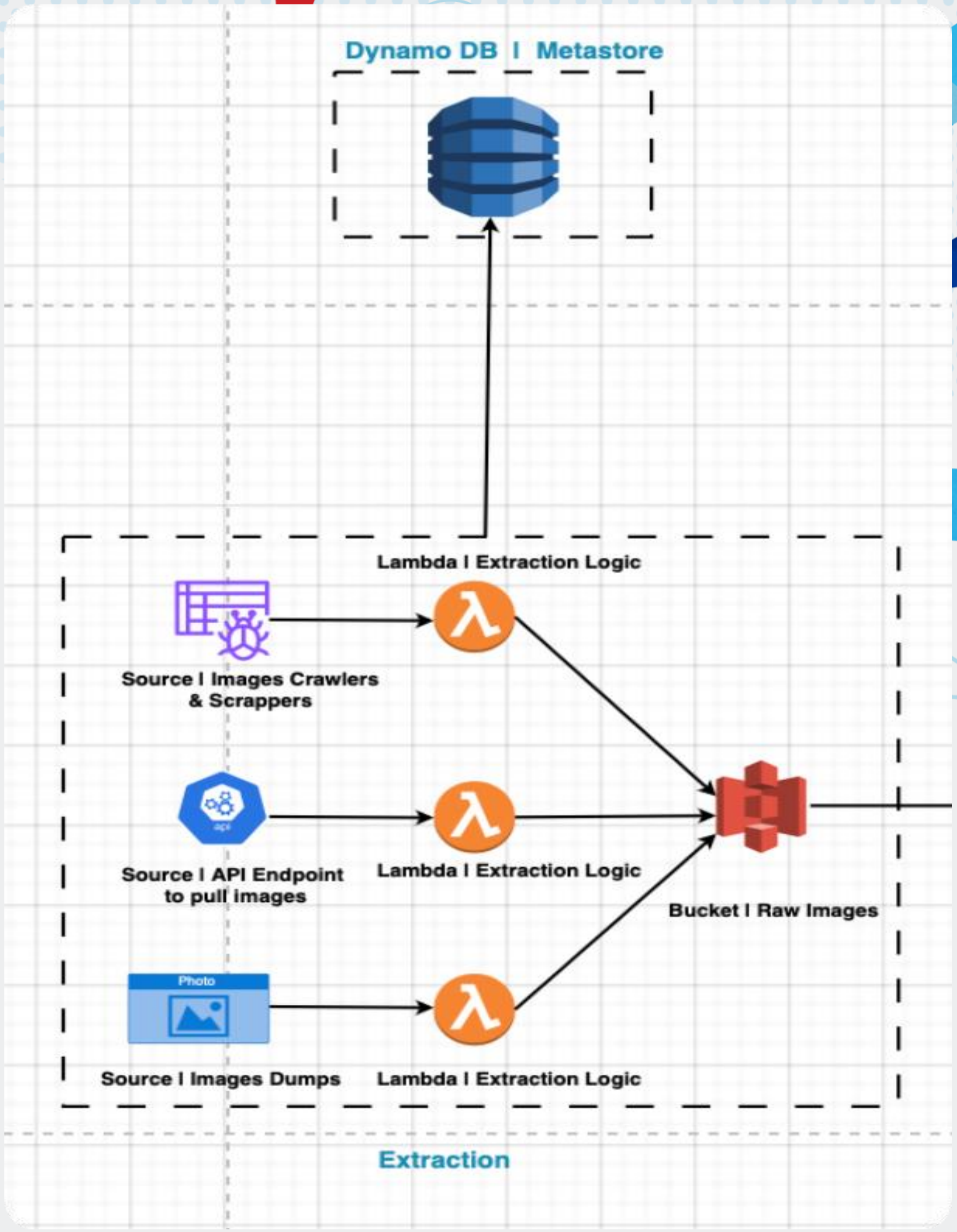
Model Results

Black and White Image

Colorized image inferred from GAN

Metastore and Data Governance

- Once the Extract lambda is triggered for a batch of Data points(images) , Lambda function also registers the Image Source and its metadata in AWS Dynamo DB Metastore.
- The approach is helpful in terms of Data Governance and further analytics about the source of Data.

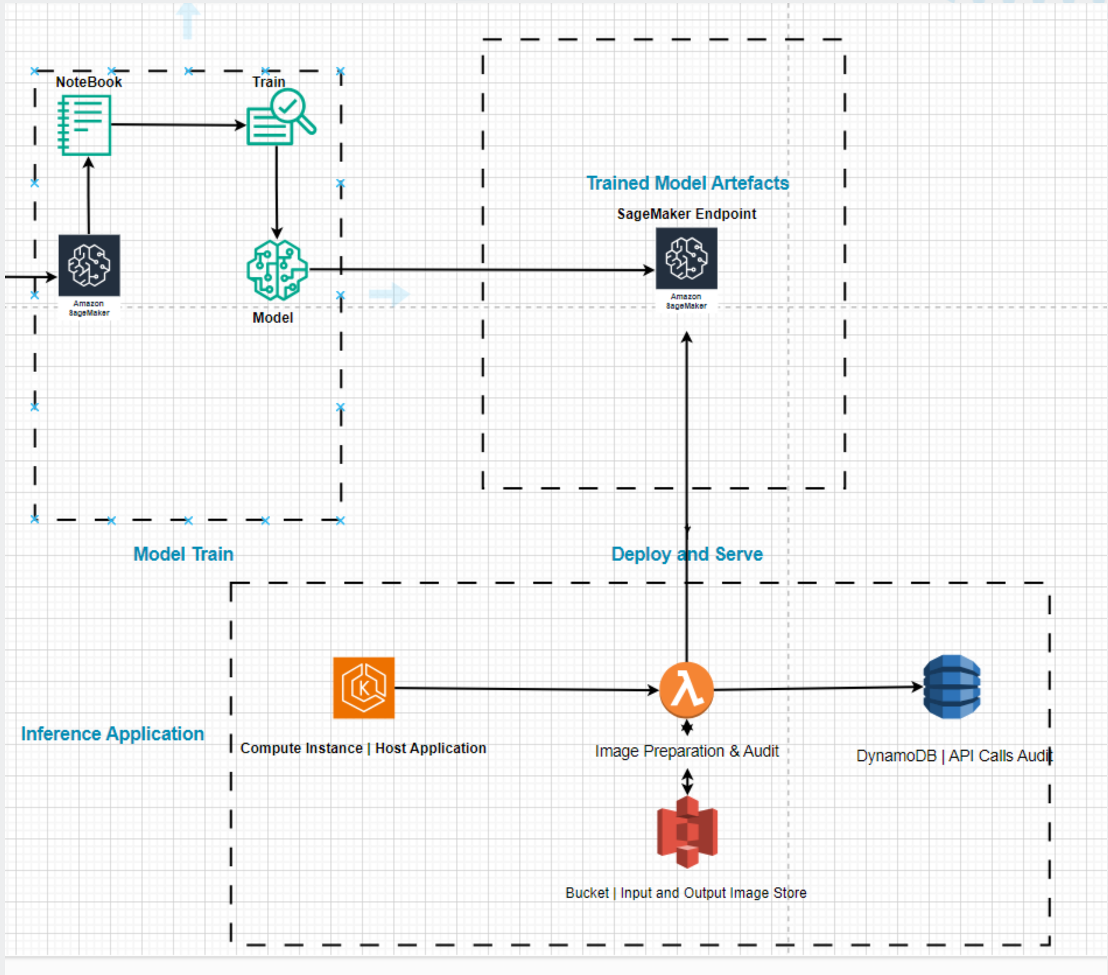


Inference Application

- Application hosted on Kubernetes Engine (Amazon EKS) using ReactJS

- Audit API requests in DynamoDB

- Store input image requests and output from model in S3 bucket for future reference.



References on Conditional GAN

https://ieeexplore.ieee.org/document/10140749

https://ieeexplore.ieee.org/document/9943130

https://gist.github.com/bonlime/4e0d236cf98cd5b15d977dfa03a63643

Thank You!