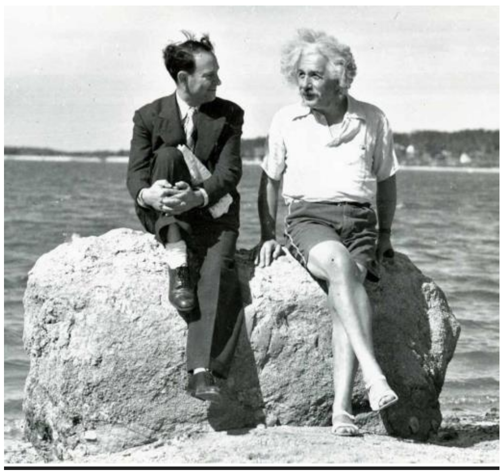
IMAGE COLORIZATION



Submitted By:
COGNITIVE THINKERS
Anuja Jain
Shatawari Jain
Jayati Kaul
Sithara Vanmerimeethal Paleri

Table of Contents

Introduction

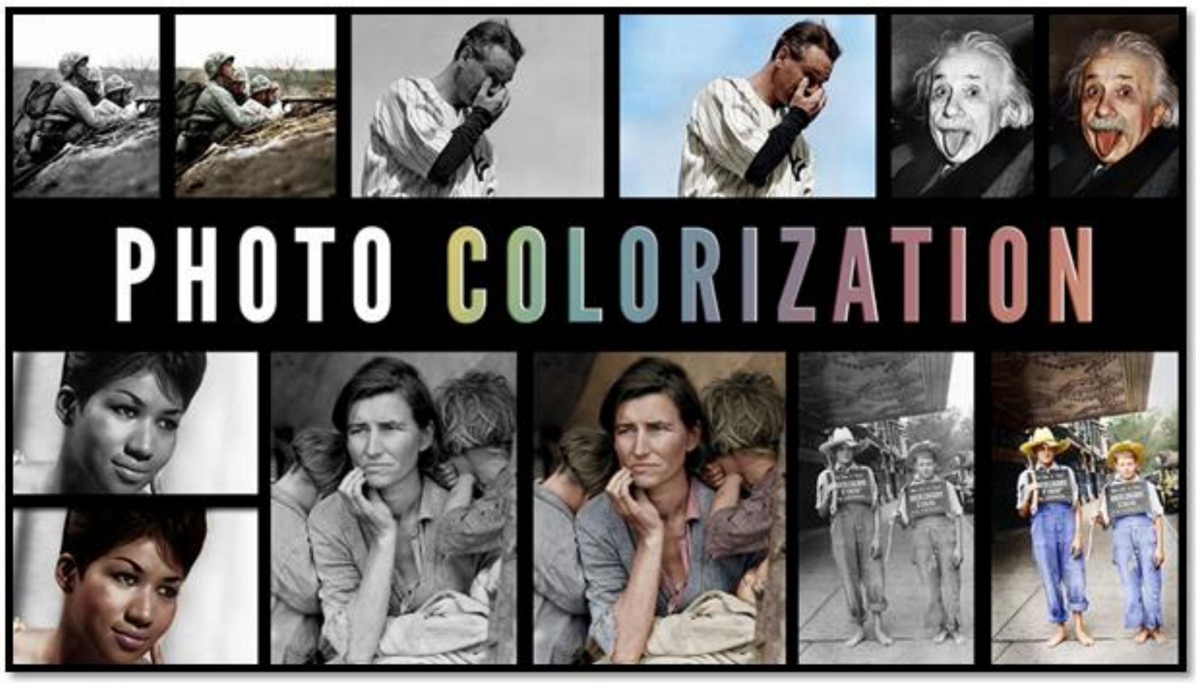


Image Analysis and Colorization
Since there are no standardized points of data in an image it is considered as unstructured data and working on unstructured data is a bit challenging task. Hitherto, we have seen a lot of developmental work on image analysis in the recent past. From object identification in an image, to converting images to texts the technology has come a far way. The main technology used behind image analysis is deep learning.

Image colorization is the process of taking an **input grayscale (black and white) image** and then producing an **output colorized image** that represents the semantic colors and tones of the input (for example, an ocean on a clear sunny day must be plausibly “blue” - it can’t be colored “hot pink” by the model).

Some of the old approaches to black and white image colorization relied on manual human annotation and they often produced desaturated results that were not “believable” as true colorizations. Previous methods for image colorization either:

- Relied on significant human interaction and annotation
- Produced desaturated colorization

The goal behind this project is to build and implement an automated technique that can be used to generate colorized images from black and white (b/w) pictures (requiring minimal manual effort). The approach that we are planning to use here is deep learning neural network. In neural networks, Convolutional neural network also known as (ConvNets or CNNs) is one of the main categories to do image recognition, image classifications, Objects detections, recognition faces etc. We are planning to utilize CNN to produce colored images.

Business Prospects

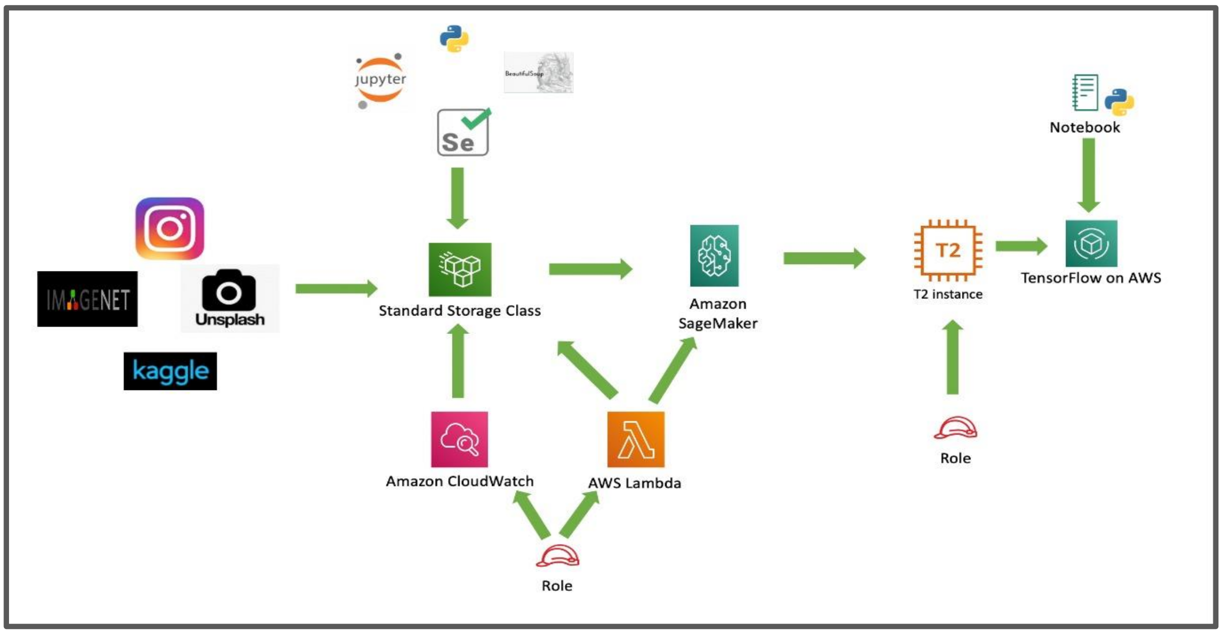
Business Problem

The goal behind our project is to transform a grayscale image into a colored image. But the question is how this will help in business and what applications will be benefitted. Some of them are listed as below -

- Investigation - When there is a need to go back to old cases, to investigate old evidence or drawing relationships between new cases and old cases, it becomes difficult if black and white images are available. Solution to this is changing them into colored photos for clear investigation. This would
- Oceanography - To create ocean images deep inside the oceans for better understanding of marine life. The absence of light deep inside the ocean makes it difficult to understand the life under. This technique will help to an extent to create an image which gives a color to the life present under there.
- Movie Industry - This would really be very useful for this industry; it will even have a lot of monetary benefits. It would really be great if we can have all the classic movies converted in colored versions.
- Reliving Memories - This would be a great opportunity to relive the old memories of your parents and grandparents. This was one of the major reasons why we chose this topic.

Architecture

Below is the architecture designed for our application.



The architecture has three main parts which connect each other-

- Data Sourcing
- Data Scraping
- AWS Services / Platform

Data has been extracted mainly from four very popular sites for image.

- Instagram
- Imagnet
- UnSplash
- Kaggle

Data Extraction is done by using Selenium and Beautiful Soap drivers in a python notebook within anaconda framework and the scraped data is pushed into AWS Standard Storage(S3) bucket. Different hashtags are used for scraping the data.

AWS Services / Platform

In this project we are using AWS platform and services to host and to build our machine learning model. We are using services like

- Cloudwatch
- Lambda
- AWS Sagemaker
    - Instance
    - Notebook
- Standard Storage system

In the first phase of the project we mainly implemented two services in the AWS platform, that is S3 and Amazon Sagemaker. In the final preparation we incorporated cloudwatch and lambda services for better understanding the data used in creating the model.

Detailed description on S3 has been mentioned in the data storage portion in the coming section. Here we will talk about the other services which we are incorporating within the platform for our project.

Amazon CloudWatch

It is a monitoring service for AWS cloud resources and the applications you run on aws. We can use cloudwatch to collect and track metrics, collect and monitor log files, set alarms and automatically react to any change in aws resources. Here we will monitor the sagemaker instance flow log and monitor the s3 bucket API calls.

- Daily storage metrics for buckets - Monitor bucket storage, which collects and processes storage data from S3 into readable, daily metrics. These storage metrics for S3 are reported once per day and are provided to all customers at no additional cost.
- Request metrics - Monitor S3 requests to quickly identify and act on operational issues. The metrics are available at 1-minute intervals after some latency to process. These CloudWatch metrics are billed at the same rate as the Amazon CloudWatch custom metrics. When enabled, request metrics are reported for all object operations. By default, these 1-minute metrics are available at the S3 bucket level.

Following are the metrics we are planning to use in our application -

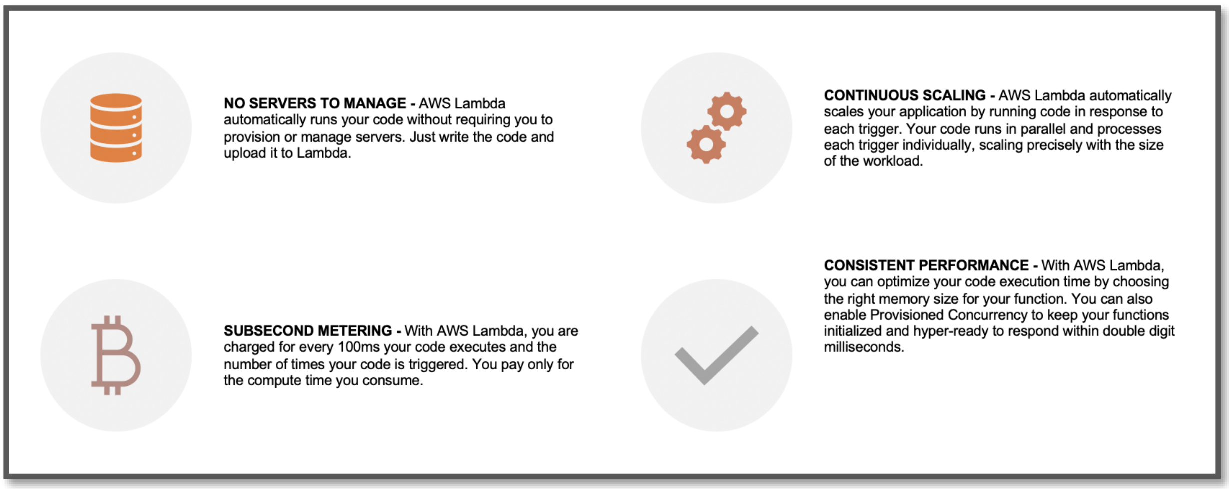
- PutRequests - The number of HTTP PUT requests made for objects in an Amazon S3 bucket.
- GetRequest - The number of HTTP GET requests made for objects in an Amazon S3 bucket. This does not include list operations.

AWS Lambda

Lambda is a Faas product. Functions are code, which runs in a runtime. Functions are invoked by events, perform actions for upto 15 minutes, and terminate. Functions are also stateless - each run in clean. We are planning to incorporate lambda into our application for the automation process, Firstly, to scrap the data in a scheduled time and also when an object falls into an s3 bucket very recently to notify the sagemaker instance to start the application.

AWS Lambda event trigger is created on S3 bucket to automatically push and transform data. Execution policy in JSON format is updated in order to assign privacy policies to Lambda Function

Benefits of using Lambda -



NO SERVERS TO MANAGE - AWS Lambda automatically runs your code without requiring you to provision or manage servers. Just write the code and upload it to Lambda.

CONTINUOUS SCALING - AWS Lambda automatically scales your application by running code in response to each trigger. Your code runs in parallel and processes each trigger individually, scaling precisely with the size of the workload.

SUBSECOND METERING - With AWS Lambda, you are charged for every 100ms your code executes and the number of times your code is triggered. You pay only for the compute time you consume.

CONSISTENT PERFORMANCE - With AWS Lambda, you can optimize your code execution time by choosing the right memory size for your function. You can also enable Provisioned Concurrency to keep your functions initialized and hyper-ready to respond within double digit milliseconds.

Amazon Sagemaker

Amazon SageMaker is a cloud machine-learning platform that was launched in November 2017. SageMaker enables developers to create, train, and deploy machine-learning models in the cloud. It also enables developers to deploy ML models on embedded systems and edge-devices.

Amazon SageMaker enables developers and data scientists to quickly and easily build, train, and deploy ML models at any scale. SageMaker provides a hosted Jupyter notebook environment that one can use to visualize and analyze data. Jupyter is an open-source web application that allows you to create visualizations and narrative text, as well as perform data cleaning, data transformation, numerical simulation, statistical modeling, and data visualization and even create custom build machine learning models.

We are using jupyter with python as a medium of language to build our model and train the data. We found out that having an ec2 instance for handling ML models is much more expensive than using a sagemaker instance. Amazon SageMaker deploys your model on an auto-scaling cluster of Amazon ec2 instances that are spread across multiple availability zones to deliver both high performance and high availability Moreover Amazon SageMaker takes away the heavy lifting of machine learning, so one can build, train, and deploy machine learning models quickly and easily.

IAM - Identity Access Management

In order to utilize better services by every member of the team, we created IAM users under the main root user and gave different username and password. This will allow us to monitor the cost of the overall account under the root user with any management difficulties. Four IAM users were created excluding the root user and each user was given full admin access to all the services for smooth functioning of the services.

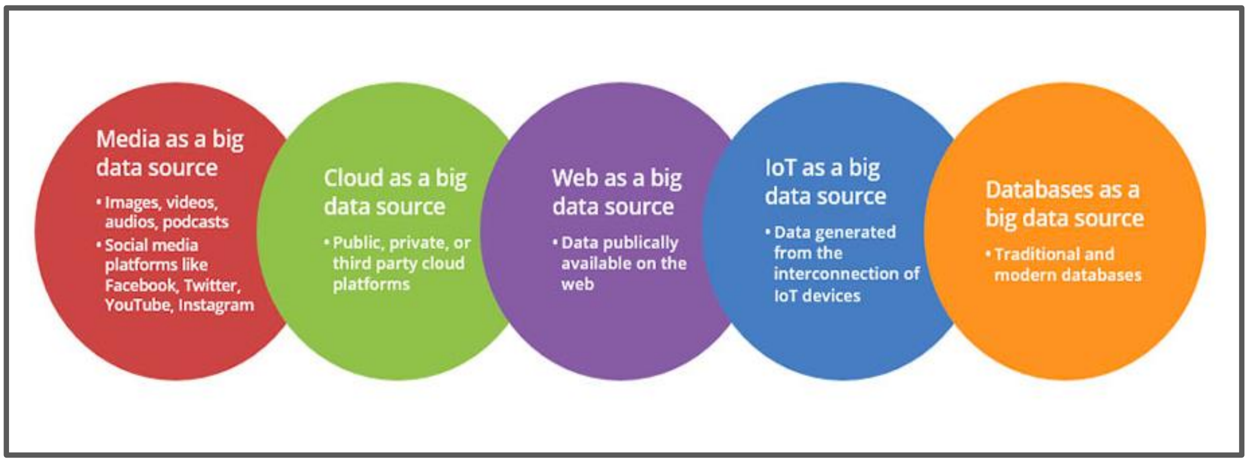
Each service in the AWS needs permission to get access to and from the users/services which are using it. There comes the service called roles which is part of IAM, which gives access to different services. An IAM role is similar to an IAM user, in that it is an AWS identity with permission policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session.

Here we have given a role which is attached to the machine learning instance, lambda, cloudwatch and the S3. We added the policy which is a Json script.

Data Sourcing

A data source is where that data is originating that will be used to run a report or gain insights. Database is the source for a database management system. Spreadsheet, XML file, data sheet or hard-coded data within the program are the data sources for computer programs. Data sources will differ depending on the computer system or program. The bulk of big data generated comes from three primary sources: social data, machine data and transactional data.

These are the top 5 sources of big data:


Media as a big data source
• Images, videos, audios, podcasts
• Social media platforms like Facebook, Twitter, YouTube, Instagram
Cloud as a big data source
• Public, private, or third party cloud platforms
Web as a big data source
• Data publically available on the web
IoT as a big data source
• Data generated from the interconnection of IoT devices
Databases as a big data source
• Traditional and modern databases

For the purpose of our project, we have considered following media/web as our data sources:

- Imagenet
- Instagram
- Unsplash
- Kaggle

ImageNet and Unsplash are online databases which provide beautiful, free images and photos. Instagram is an American photo and video-sharing social networking service owned by Facebook, Inc. Kaggle, a subsidiary of Google LLC, is an online community of data scientists and machine learning practitioners.

Imagenet - ImageNet is an image database organized based on the WordNet hierarchy currently they only have nouns, in which each node of the hierarchy has a collection of hundreds and thousands of images. Currently, they have an average of over five hundred images per node. ImageNet is a useful resource for researchers, educators, students and others which need a large collection of images. We have extracted data using filter criteria. We collected around 1000 images from this source

Instagram - Instagram is an American photo and video-sharing social networking service owned by Facebook, Inc. Using different hashtags, we were able to scrape around 2000 images from Instagram

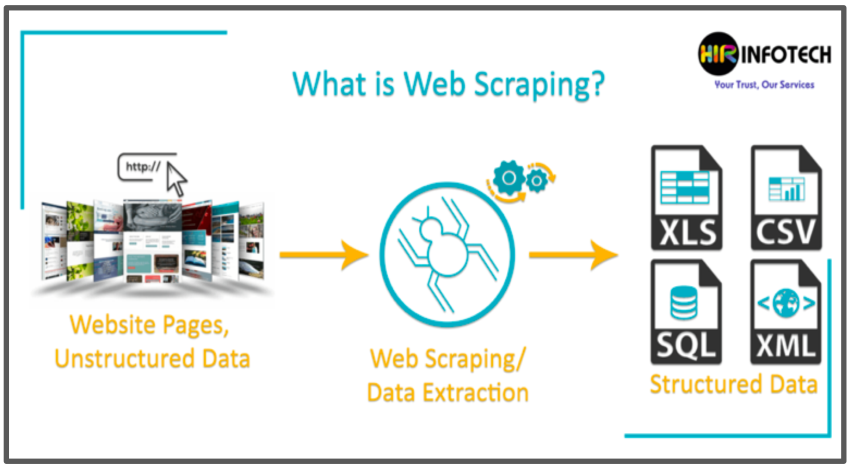
Unsplash - Unsplash is a website for sharing photography under the Unsplash permit. The site claims that they have more than 110,000 contributing photographers and generates more than 11 billion photographs impressions for each month on their library of over 1.5 million photographs. We used search keywords to collect images from this website. We collected around 1000 images from this.

We used the above-mentioned websites to collect images using different search keywords/hashtags and different filter criteria in order to collect different kinds and flavors of images for training our model. The total extracted data summed up to 4 GB to train our model.

Kaggle - Kaggle is an online platform for data scientists and machine learning practitioners. We have nearly 3GB of data to train our model. There are two datasets available for this project and both are in .nyp format. One dataset is a gray_scale.npy file which is the grayscale version of the MIRFLICKR25k dataset and the other one contains 25 .npy files consisting of a and b dimensions of LAB color space images.

Data Scraping

Web scraping is a technique for extracting information from the internet pages automatically using a software. It helps us extract large volumes of data about customers, products, people, stock markets, etc. It is usually difficult to get this kind of information on a large scale using traditional data collection methods. Also, it will take time to scrape the images manually, so this increases the efficiency as well.

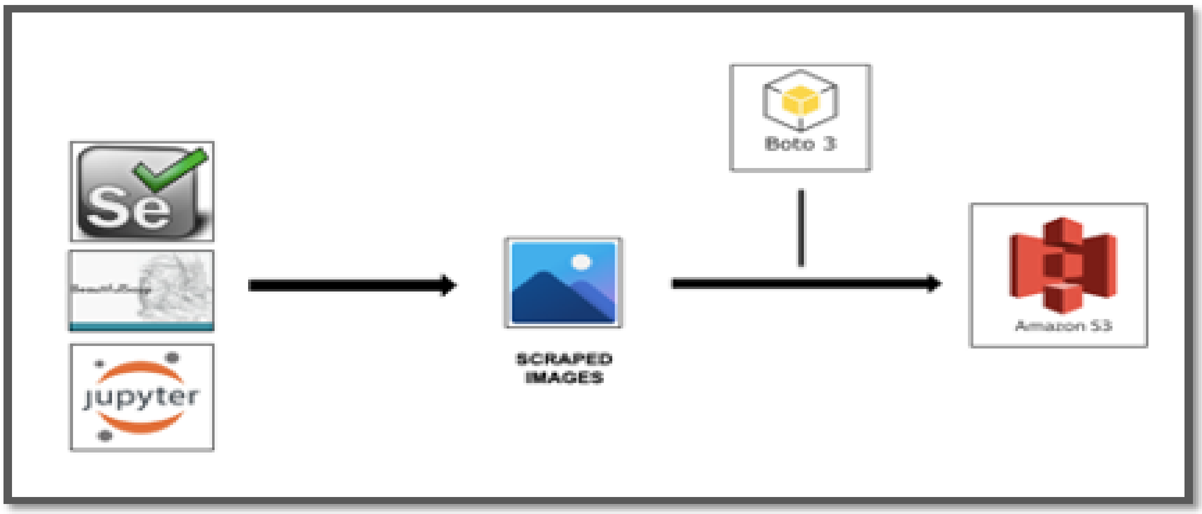


There are many uses for Web Scraping, but we will mention just a few:

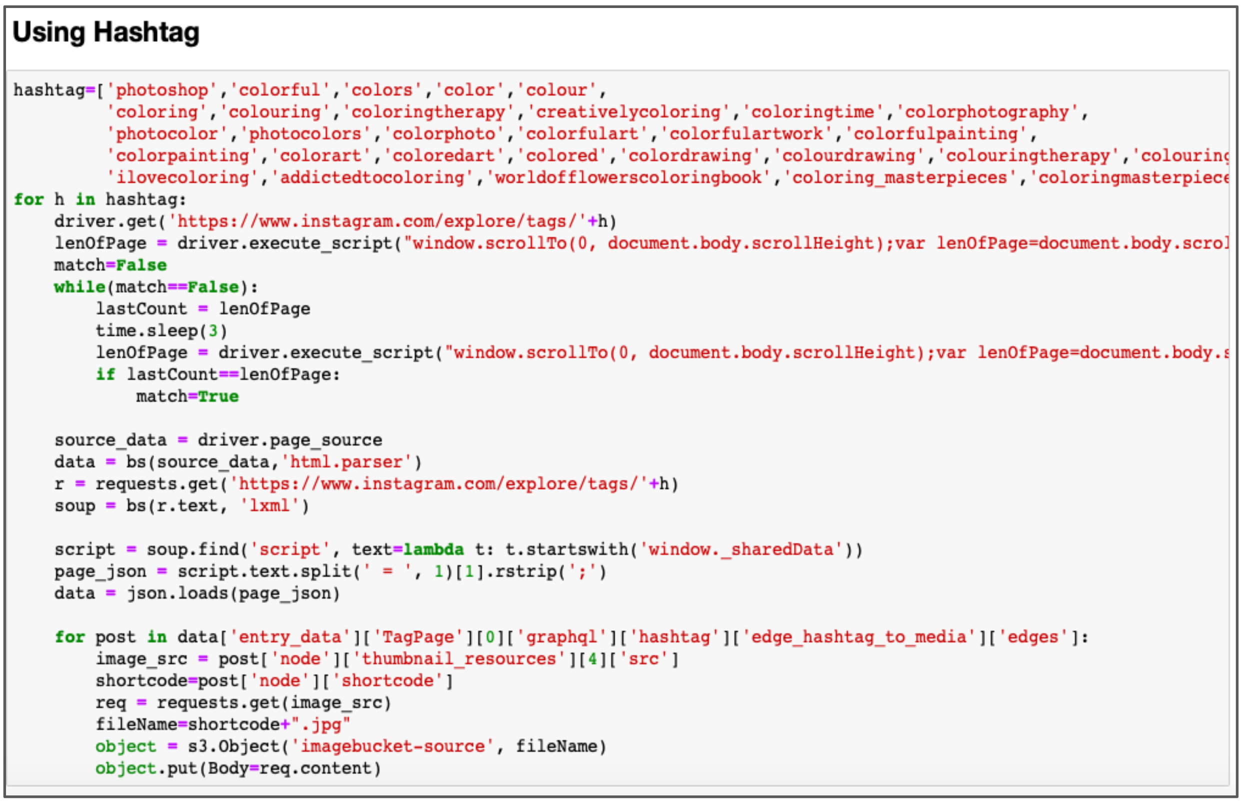
- Contact Scraping
- Data Mining
- Online Price Change Monitoring & Price Comparison
- Product Review Scraping: to watch your competition
- Gathering Real Estate Listings
- Weather Data Monitoring
- Website Change Detection
- Research
- Tracking Online Presence and Reputation
- Web Data Integration

In our project, we are doing it in Jupyter notebook. We are using Selenium WebDriver that drives the browser using browser's built-in support. Also, to parse the source page, we are using beautiful soup. Beautiful Soup is a Python package for parsing HTML and XML documents. We are using Boto3 to store images in S3.

Boto is the Amazon Web Services (AWS) SDK for Python. It enables Python developers to create, configure, and manage AWS services, such as EC2 and S3. Boto provides an easy to use, object-oriented API, as well as low-level access to AWS services.



We go through the body of the html script and extract the display_url for each image in that page and we store the images in S3 bucket. We used different hashtags as filters to get the images from the sources. They are stored in the bucket called imagebucket-source.This is being triggered manually and would like to automate it using Lambda.


Using Hashtag

```python
hashtag=['photoshop','colorful','colors','color','colour',
         'coloring','colouring','coloringtherapy','creativelycoloring','coloringtime','colorphotography',
         'photocolor','photocolors','colorphoto','colorfulart','colorfulartwork','colorfulpainting',
         'colorpainting','colorart','coloredart','colored','colordrawing','colourdrawing','colouringtherapy','colourin
         'ilovecoloring','addictedtocoloring','worldofflowerscoloringbook','coloring_masterpieces','coloringmasterpiece
for h in hashtag:
    driver.get('https://www.instagram.com/explore/tags/'+h)
    lenOfPage = driver.execute_script("window.scrollTo(0, document.body.scrollHeight);var lenOfPage=document.body.scrol
    match=False
    while(match==False):
        lastCount = lenOfPage
        time.sleep(3)
        lenOfPage = driver.execute_script("window.scrollTo(0, document.body.scrollHeight);var lenOfPage=document.body.s
        if lastCount==lenOfPage:
            match=True

    source_data = driver.page_source
    data = bs(source_data,'html.parser')
    r = requests.get('https://www.instagram.com/explore/tags/'+h)
    soup = bs(r.text, 'lxml')

    script = soup.find('script', text=lambda t: t.startswith('window._sharedData'))
    page_json = script.text.split(' = ', 1)[1].rstrip(';')
    data = json.loads(page_json)

    for post in data['entry_data']['TagPage'][0]['graphql']['hashtag']['edge_hashtag_to_media']['edges']:
        image_src = post['node']['thumbnail_resources'][4]['src']
        shortcode=post['node']['shortcode']
        req = requests.get(image_src)
        fileName=shortcode+".jpg"
        object = s3.Object('imagebucket-source', fileName)
        object.put(Body=req.content)
```

Data Storage

The most popular and widely used storage in AWS is S3 also known as Simple Storage Service. S3 is a globally available service and can be accessed from the AWS Management console, which is a simple and intuitive web interface.

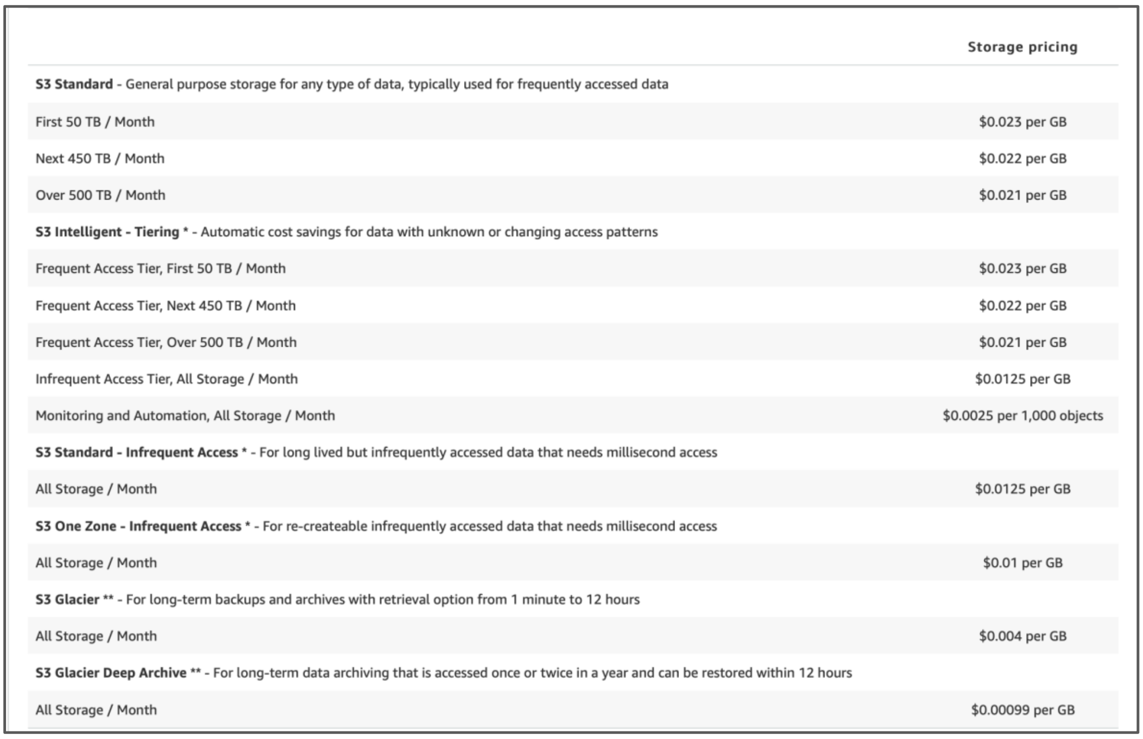
To understand better we need to look closely into some simple concepts. Amazon S3 stores the data as objects within buckets. An object can consist of files or any metadata that describes that file. There are certain steps to follow to store an object in S3, when you upload a file you can set certain permissions over the bucket as well as for the file.

Buckets are the containers for the objects. You can create more than one bucket.
By default, users can provision up to 100 buckets per AWS account. However, you can increase your Amazon S3 bucket limit by visiting AWS Service Limits. An object can be 0 bytes to 5TB. For objects larger than 100 megabytes, users should consider using the Multipart Upload capability.

There are different storage spaces available in S3 depending on the user’s requirement. The following are the different storage spaces and features-

- Standard S3 (General Purpose)
- S3 intelligent - Tiering
- Infrequent Access
- S3 One Zone-Infrequent Access (S3 one Zone-IA)
- S3 Glacier / S3 Glacier Deep Archive

Below mentioned diagram specifies the costing involved with these different storage types -



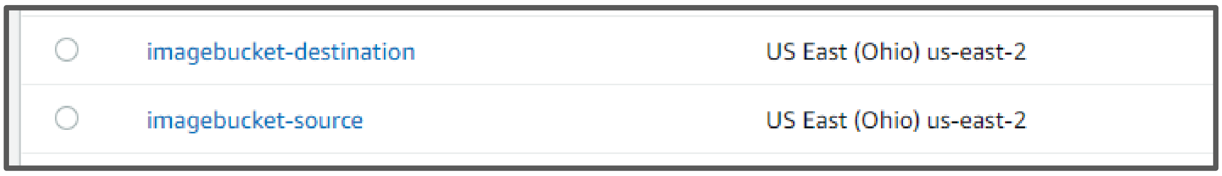
	Storage pricing
S3 Standard - General purpose storage for any type of data, typically used for frequently accessed data	
First 50 TB / Month	$0.023 per GB
Next 450 TB / Month	$0.022 per GB
Over 500 TB / Month	$0.021 per GB
S3 Intelligent - Tiering * - Automatic cost savings for data with unknown or changing access patterns	
Frequent Access Tier, First 50 TB / Month	$0.023 per GB
Frequent Access Tier, Next 450 TB / Month	$0.022 per GB
Frequent Access Tier, Over 500 TB / Month	$0.021 per GB
Infrequent Access Tier, All Storage / Month	$0.0125 per GB
Monitoring and Automation, All Storage / Month	$0.0025 per 1,000 objects
S3 Standard - Infrequent Access * - For long lived but infrequently accessed data that needs millisecond access	
All Storage / Month	$0.0125 per GB
S3 One Zone - Infrequent Access * - For re-createable infrequently accessed data that needs millisecond access	
All Storage / Month	$0.01 per GB
S3 Glacier ** - For long-term backups and archives with retrieval option from 1 minute to 12 hours	
All Storage / Month	$0.004 per GB
S3 Glacier Deep Archive ** - For long-term data archiving that is accessed once or twice in a year and can be restored within 12 hours	
All Storage / Month	$0.00099 per GB

We are planning to use the below mentioned two types of storage for our project and thus getting into little detailed explanation on these storage types and reasoning behind why we plan to implement them.

- Standard S3 (General Purpose)
  - High durability, availability and performance object storage for frequently accessed data
  - Designed for durability of 99.999999999% of objects across multiple availability zones
  - S3 Lifecycle management for automatic migration of objects to other S3 Storage classes
- S3 Glacier / S3 Glacier Deep Archive
  - S3 Glacier is a secure, durable, and low-cost storage class for data archiving
  - Configurable retrieval times, from minutes to hours
  - S3 Glacier Deep Archive is Amazon S3's lowest-cost storage class and supports long-term retention and digital preservation for data that may be accessed once or twice in a year
  - Retrieval time within 12 hours

In our project we are using Standard S3 storage class as it comes with the free tier AWS account and has lighting speed retrieval time. We will enable Lifecycle Management for the S3 bucket which we are using. Once the images are already used for the color conversion it is no more needed in the recent activity anymore. Therefore, through lifecycle management we will transfer the data to Glacier storage which is much cheaper than the S3 standard storage.

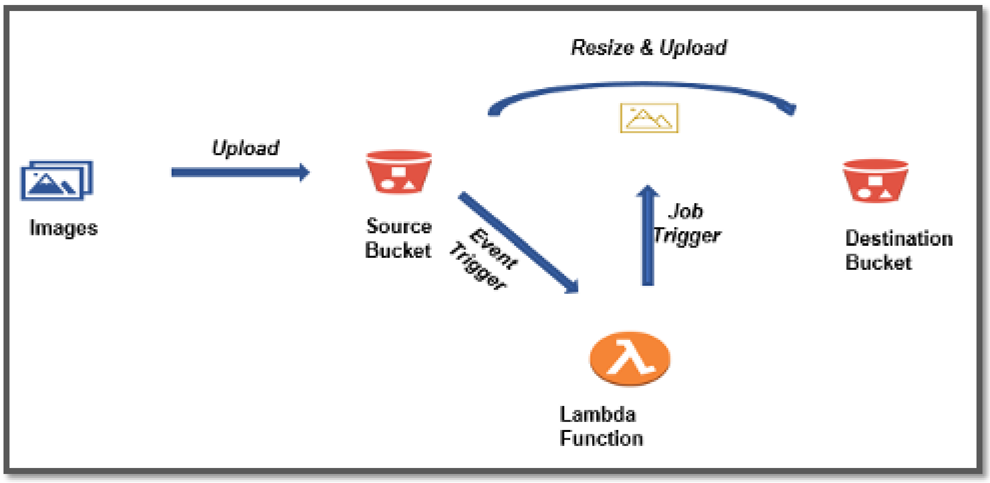
For the purpose of this project we created two S3 standard buckets as mentioned in the below diagram -



○	imagebucket-destination	US East (Ohio) us-east-2
○	imagebucket-source	US East (Ohio) us-east-2

Data Stitching and Transformation

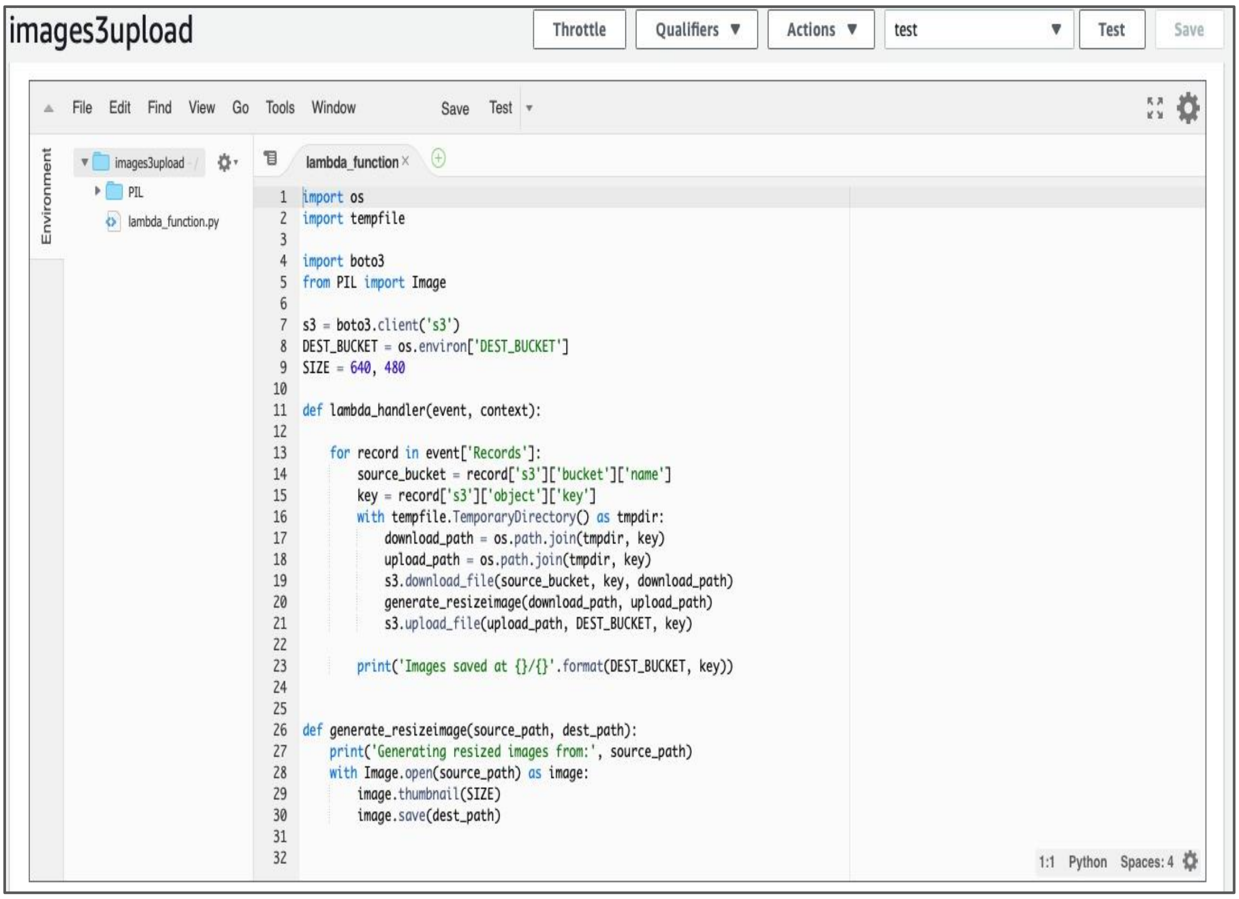
Data stitching is the process of combining your datasets from multiple sources in a way that creates a transformed dataset which can be directly used for modelling and this data transformation helps in better performance when the model is trained on this dataset.



We scraped data from multiple sources and pushed the collected data to S3 bucket called ‘imagebucket-source’. This bucket contains all the raw data as collected from the different data sources. In order to use this data for training we need to stitch and transform this data to a uniform pixel quality to produce better results from our model. All the images are transformed to uniform pixel quality with size (640,480) and the transformed images are then uploaded to S3 destination bucket called ‘imagebucket-destination’.

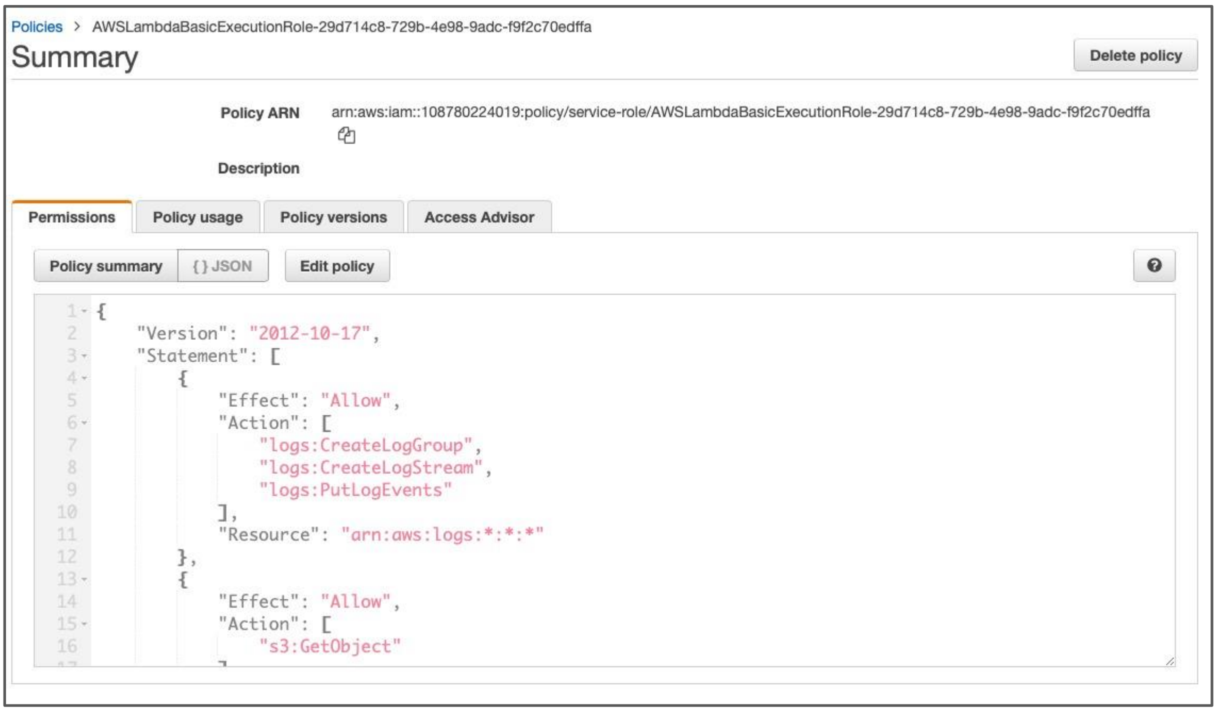
To achieve the above-mentioned data transformation, we utilized AWS Lambda service -

- AWS Lambda function is created on S3 source bucket to transform the image data and push the transformed data to destination bucket
- The Lambda function uses PIL library which is the Python Image Library that adds image processing capabilities to the Python interpreter
- Environment variables are set with their values, here destination bucket is the environment variable and its value is set as imagebucket-destination



```python
import os
import tempfile

import boto3
from PIL import Image

s3 = boto3.client('s3')
DEST_BUCKET = os.environ['DEST_BUCKET']
SIZE = 640, 480

def lambda_handler(event, context):

    for record in event['Records']:
        source_bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']
        with tempfile.TemporaryDirectory() as tmpdir:
            download_path = os.path.join(tmpdir, key)
            upload_path = os.path.join(tmpdir, key)
            s3.download_file(source_bucket, key, download_path)
            generate_resizeimage(download_path, upload_path)
            s3.upload_file(upload_path, DEST_BUCKET, key)

        print('Images saved at {}/{}'.format(DEST_BUCKET, key))


def generate_resizeimage(source_path, dest_path):
    print('Generating resized images from:', source_path)
    with Image.open(source_path) as image:
        image.thumbnail(SIZE)
        image.save(dest_path)
```

- AWS Lambda object PUT event trigger is created on S3 bucket to automatically push and transform data as and when data is uploaded on S3 source bucket
- Execution policy in JSON format is updated in order to provide access to Lambda Function to S3 source and destination buckets

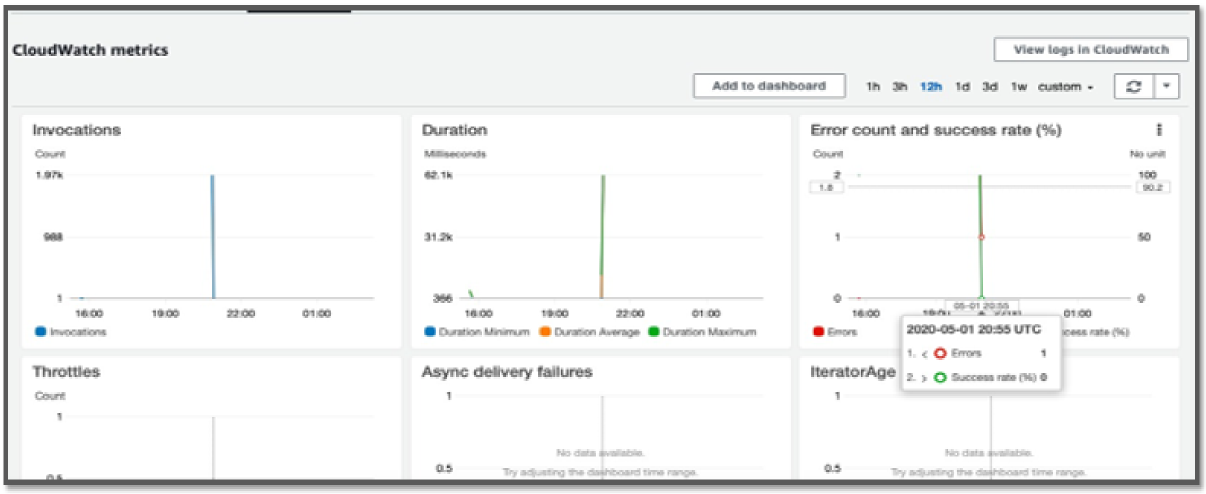


Policy ARN arn:aws:iam::108780224019:policy/service-role/AWSLambdaBasicExecutionRole-29d714c8-729b-4e98-9adc-f9f2c70edffa

Description

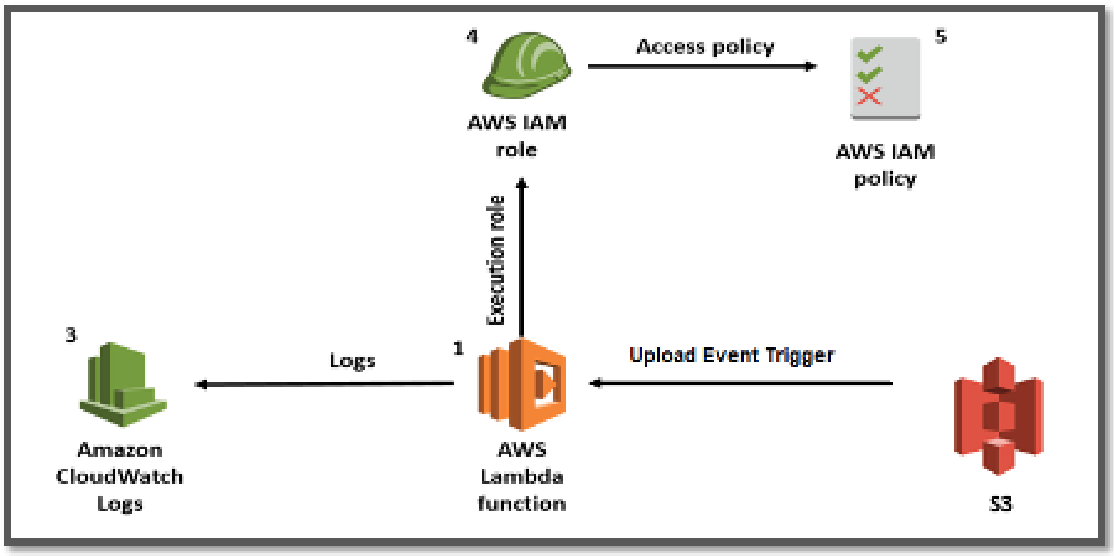
```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
```

- Monitoring can be done using Lambda function monitoring tab

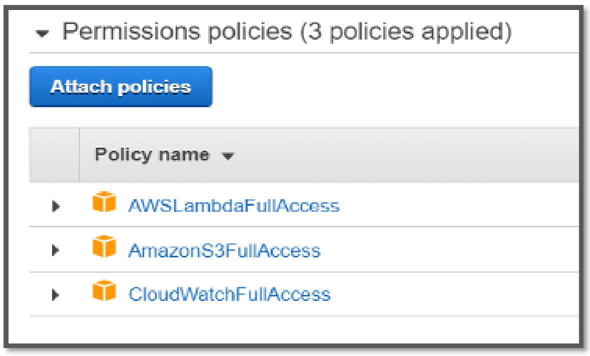


Data Monitoring

AWS Lambda can be used to integrate with other AWS services. Here, we used it to monitor and analyze the logs in Amazon CloudWatch Logs. CloudWatch collects monitoring and operational data in the form of logs, metrics, and events, providing you with a unified view of AWS resources, applications, and services that run on AWS and on-premises servers.

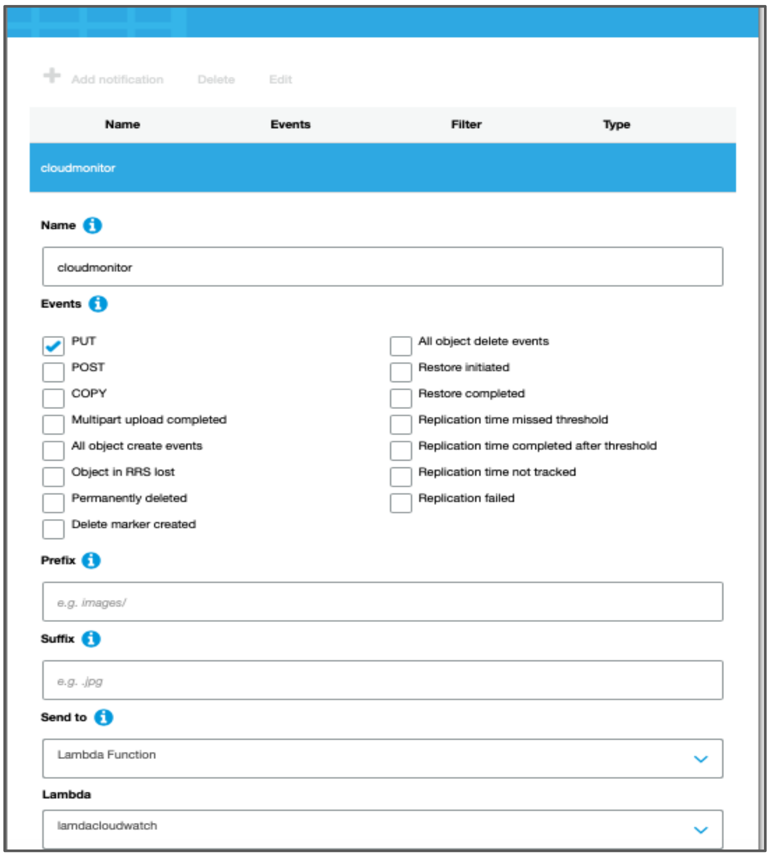


We created an IAM role and attached 3 policies which Lambda will use to work with S3 and to write logs to CloudWatch.



Policy name
AWSLambdaFullAccess
AmazonS3FullAccess
CloudWatchFullAccess

In order to create logs, we created a trigger by configuring a put event on the 'imagebucket-destination', meaning every time an image is uploaded in the bucket, it will create a log. As soon as an image hits our destination bucket, we will run our SageMaker instance manually.



Name	Events	Filter	Type
cloudmonitor			

Name: cloudmonitor

Events:

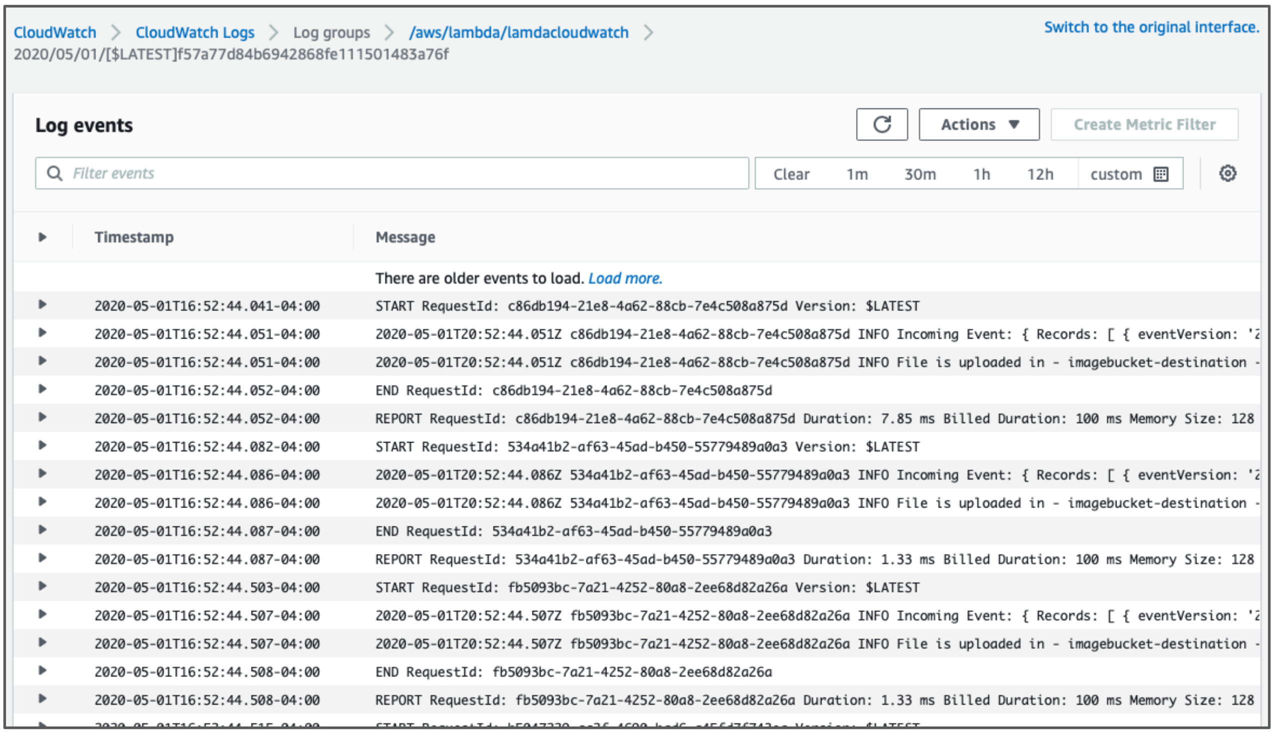
- [✓ PUT]
- POST
- COPY
- Multipart upload completed
- All object create events
- Object in RRS lost
- Permanently deleted
- Delete marker created
- All object delete events
- Restore initiated
- Restore completed
- Replication time missed threshold
- Replication time completed after threshold
- Replication time not tracked
- Replication failed

Prefix: e.g. images/

Suffix: e.g. .jpg

Send to: Lambda Function

Lambda: lamdacloudwatch

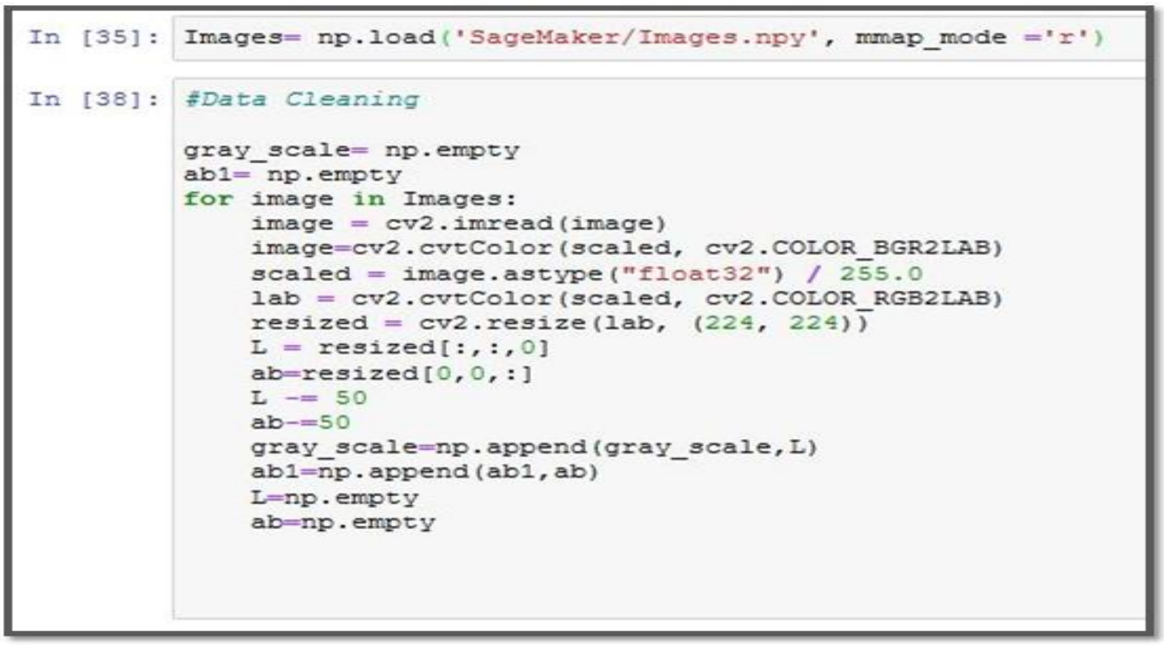

CloudWatch > CloudWatch Logs > Log groups > /aws/lambda/lamdacloudwatch > 2020/05/01/[$LATEST]f57a77d84b6942868fe111501483a76f
Switch to the original interface.
Log events
Actions
Create Metric Filter
Filter events
Clear 1m 30m 1h 12h custom

```
Timestamp                      Message
                               There are older events to load. Load more.
2020-05-01T16:52:44.041-04:00  START RequestId: c86db194-21e8-4a62-88cb-7e4c508a875d Version: $LATEST
2020-05-01T16:52:44.051-04:00  2020-05-01T20:52:44.051Z c86db194-21e8-4a62-88cb-7e4c508a875d INFO Incoming Event: { Records: [ { eventVersion: '2
2020-05-01T16:52:44.051-04:00  2020-05-01T20:52:44.051Z c86db194-21e8-4a62-88cb-7e4c508a875d INFO File is uploaded in - imagebucket-destination -
2020-05-01T16:52:44.052-04:00  END RequestId: c86db194-21e8-4a62-88cb-7e4c508a875d
2020-05-01T16:52:44.052-04:00  REPORT RequestId: c86db194-21e8-4a62-88cb-7e4c508a875d Duration: 7.85 ms Billed Duration: 100 ms Memory Size: 128
2020-05-01T16:52:44.082-04:00  START RequestId: 534a41b2-af63-45ad-b450-55779489a0a3 Version: $LATEST
2020-05-01T16:52:44.086-04:00  2020-05-01T20:52:44.086Z 534a41b2-af63-45ad-b450-55779489a0a3 INFO Incoming Event: { Records: [ { eventVersion: '2
2020-05-01T16:52:44.086-04:00  2020-05-01T20:52:44.086Z 534a41b2-af63-45ad-b450-55779489a0a3 INFO File is uploaded in - imagebucket-destination -
2020-05-01T16:52:44.087-04:00  END RequestId: 534a41b2-af63-45ad-b450-55779489a0a3
2020-05-01T16:52:44.087-04:00  REPORT RequestId: 534a41b2-af63-45ad-b450-55779489a0a3 Duration: 1.33 ms Billed Duration: 100 ms Memory Size: 128
2020-05-01T16:52:44.503-04:00  START RequestId: fb5093bc-7a21-4252-80a8-2ee68d82a26a Version: $LATEST
2020-05-01T16:52:44.507-04:00  2020-05-01T20:52:44.507Z fb5093bc-7a21-4252-80a8-2ee68d82a26a INFO Incoming Event: { Records: [ { eventVersion: '2
2020-05-01T16:52:44.507-04:00  2020-05-01T20:52:44.507Z fb5093bc-7a21-4252-80a8-2ee68d82a26a INFO File is uploaded in - imagebucket-destination -
2020-05-01T16:52:44.508-04:00  END RequestId: fb5093bc-7a21-4252-80a8-2ee68d82a26a
2020-05-01T16:52:44.508-04:00  REPORT RequestId: fb5093bc-7a21-4252-80a8-2ee68d82a26a Duration: 1.33 ms Billed Duration: 100 ms Memory Size: 128
```

Data Cleaning

Data cleaning is a process of finding out and correcting inaccurate records from a database and it also means to identify incomplete, incorrect, inaccurate parts of data and then replacing or modifying or deleting that part.
As we know that for any kind of data analysis, data cleaning is an important task to achieve. With images also it is the same. Since the information stored in images is in pixel values, when we load or read an image into python, the pixel values of an image are stored in an array. But even that array has non-uniform data (images might be of different quality), so to make the data uniform and to have the CNN model take it as an input we executed the following steps:

- Scaled the pixel intensities to the range [0, 1]
- Converted images from RGB to LAB color model
- Resized the Lab image to 224x224 (the dimensions which the colorization network accepts)
- Compressed the image into a 2-D matrix (NumPy array)



```
In [35]: Images= np.load('SageMaker/Images.npy', mmap_mode ='r')

In [38]: #Data Cleaning

         gray_scale= np.empty
         ab1= np.empty
         for image in Images:
             image = cv2.imread(image)
             image=cv2.cvtColor(scaled, cv2.COLOR_BGR2LAB)
             scaled = image.astype("float32") / 255.0
             lab = cv2.cvtColor(scaled, cv2.COLOR_RGB2LAB)
             resized = cv2.resize(lab, (224, 224))
             L = resized[:,:,0]
             ab=resized[0,0,:]
             L -= 50
             ab-=50
             gray_scale=np.append(gray_scale,L)
             ab1=np.append(ab1,ab)
             L=np.empty
             ab=np.empty
```

Proposed Solution

Based on our problem statement it is quite understandable that many industries as mentioned above, will be benefited if automated Image Colorization of black and white images can be implemented. This problem is yet challenging because it is multimodal which means a single grayscale image may correspond to many plausible colored images. And as per our research and understanding we believe that using deep learning neural networks, we would be able to achieve the desired solution. CNN is one of the deep learning neural networks which we are planning to utilize for achieving a solution to our Business Problem.

What is CNN

Convolutional neural network also known as ConvNets or CNNs is one of the main techniques to do images recognition, images classifications, Objects detections, recognition faces etc., It can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. This algorithm does not require much preprocessing. While in primitive methods filters used are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

CNN Applications

CNNs are widely used and they can be found at the core of everything from Facebook's photo tagging to self-driving cars. Some of the applications on CNNs are -

- Image recognition and OCR - CNNs use Optical Character Recognition (OCR) to classify and cluster peculiar elements like letters and numbers.
- Object detection for self-driving cars - Convolutional Neural Networks (CNNs) are widely used to detect objects for self-driving cars
- Face recognition on social media - Deep learning-based methods have shown better performances in terms of accuracy and speed of processing in image recognition utilized by almost all social media platforms
- Image analysis in healthcare - Even in healthcare image analysis is of great importance and CNN medical image classification can detect the anomalies on the X-ray or MRI images with higher precision than the human eye.

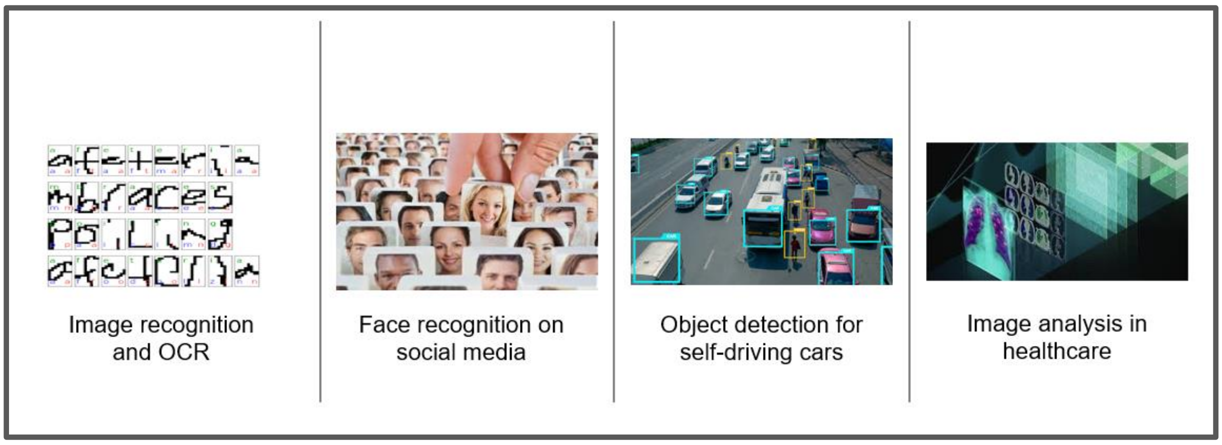


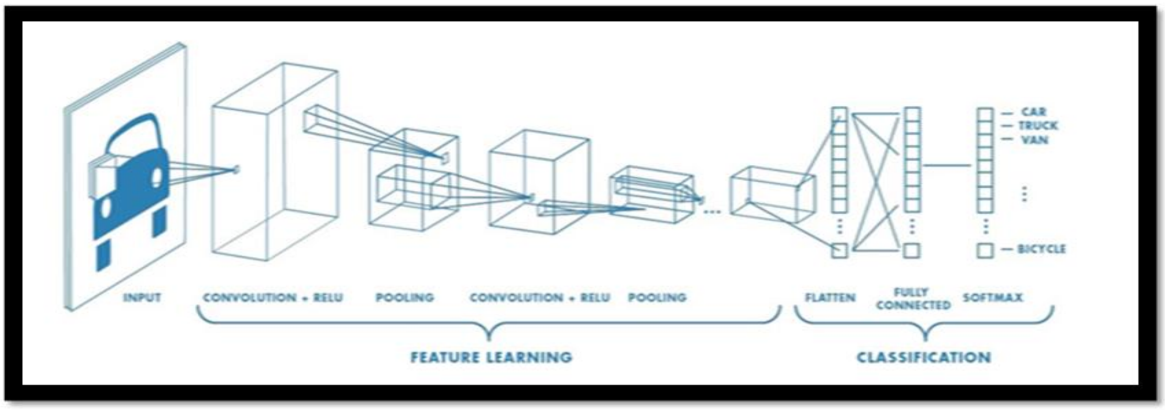
Image recognition and OCR
Face recognition on social media
Object detection for self-driving cars
Image analysis in healthcare

A CNN works by extracting features from images and in this model the features are not trained instead they are learned while the network trains on a set of images.

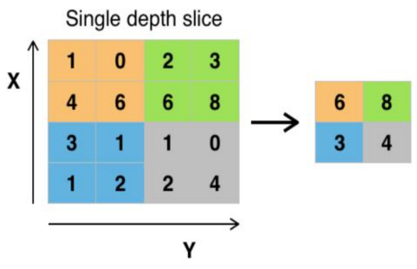
Architecture of CNN

The architecture of a ConvNet is very similar to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the association of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.

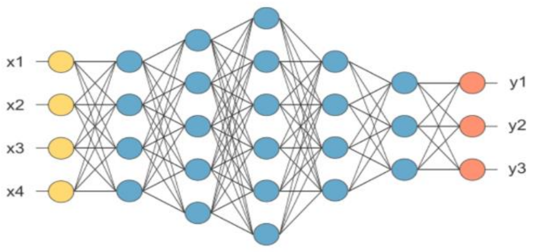
CNNs have an input layer, and output layer, and hidden layers. The hidden layers consist of convolutional layers, ReLU layers, pooling layers, and fully connected layers.



- Convolutional layers - It receives input and then transforms the input and passes it onto the next layer. For each convolutional layer we decide the number filters the layer should have and these filters detect patterns which include edges, shapes, textures, object etc. Filters or we might be familiar with terms like neurons and kernel are matrix which are initialized by random numbers and then the images convolve against each filter
- ReLu Layer - Then the output goes to the ReLu layer which is the Rectified Linear Unit layer. This layer is an extension of a convolutional layer. The purpose of this layer is to increase the non-linearity of the image. And this is done in order to provide a better feature extraction. According to the researchers, ReLU layers work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy. It also helps to alleviate the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers
- Pooling layer - We may choose to apply or not apply this layer. It is also called a downsampling layer. It is designed to reduce the number of parameters of the input images. This basically takes a filter (normally of size 2x2) and a stride of the same length. It then applies it to the input volume and outputs the maximum number in every subregion that the filter convolves around.



- Fully Connected Layer - The matrix is then flattened into a vector and fed into Fully connected layer like a neural network. This layer takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to



choose from and then determines which features most correlate to a particular class. Then, we have an activation function which classifies the outputs.

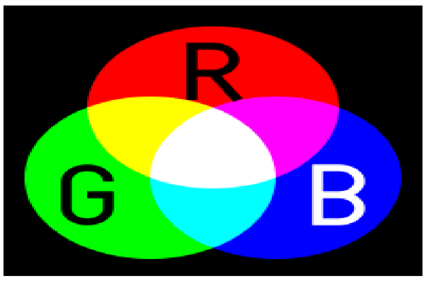
This was a brief introduction on how CNN is used in Image classification. Now to have insights on Image Colorization using CNN we first need to understand **color space.**

Color Space

In this topic we would like to elaborate about two color models namely RGB and Lab. The reason we are discussing these models is because it is pivotal in understanding the deep learning technique we are going to use in our project.

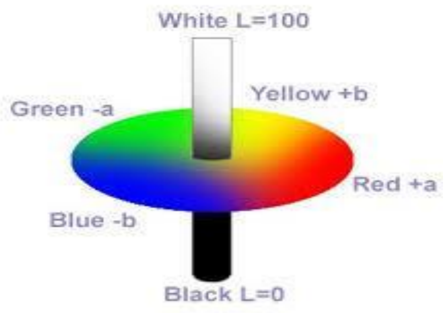
RGB Color Model

It is an additive model made by the addition of the three colors **red, green** and **blue** in various ways to produce a broad array of colors. This model's main purpose is for sensing, displaying and representation of images on electronic sources such as television, mobile and laptop. Also, it is used for developing conventional photographs.



CIELab Model (L*a*b)

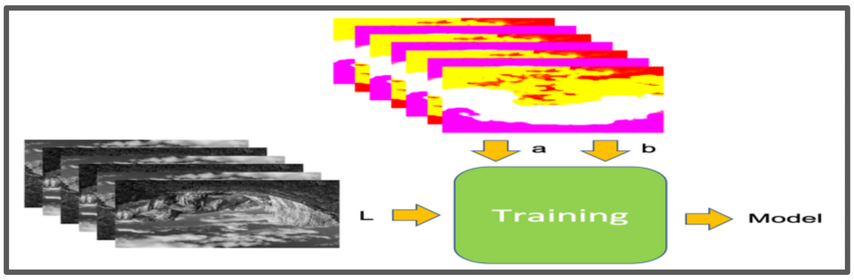
This color model is based on the theory of opposing colors. Its underlying principle is that a color cannot be green and red or yellow or blue at the same time. It is called Lab color model usually and has 3 channels for depicting the colors and lightness. The **L** channel stands for lightness intensity (value ranges from 0(black)-100(white)), **a** channel stands for value of color from green (-a) to red (+a) and the **b** channel stands for value of color from blue(-b) to yellow (+b). Since lightness intensity is even present in grayscale images, it will be easier to use this color model in training and prediction rather than the RGB color model.



Modelling

Image Colorization Technique
Here, we will be explaining the technique of model training and prediction used in our project for image colorization.

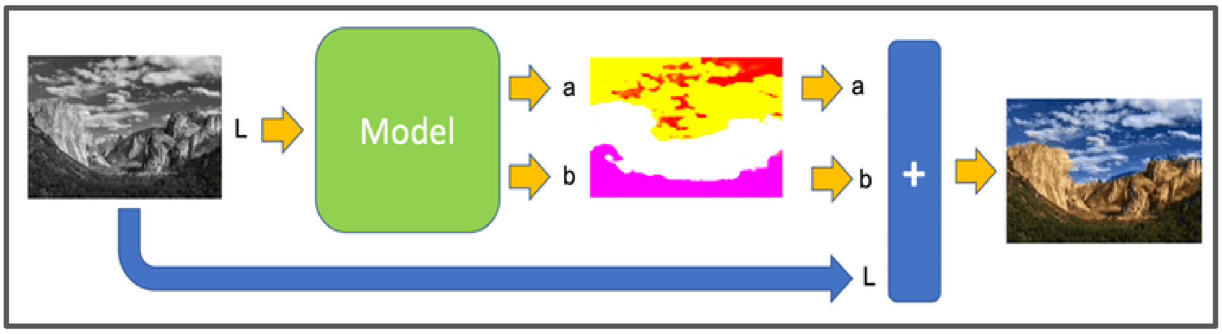
Model Training



After researching we found out that the technique used for image colorization involves the following steps for training the model:

- Convert all the training images in RGB color space to Lab color space
- Using the L part of the input image to the network and training the network to predict the a and b channels of Lab
- Combining the input L channel with the predicted ab channels to make a synthesized Lab image.
- Converting the Lab image back to RGB

Model Prediction



After training the model and tuning it, we will be using the trained model for prediction i.e. giving a grayscale image as an input and getting a colorized image as an output. The method involves the following steps:

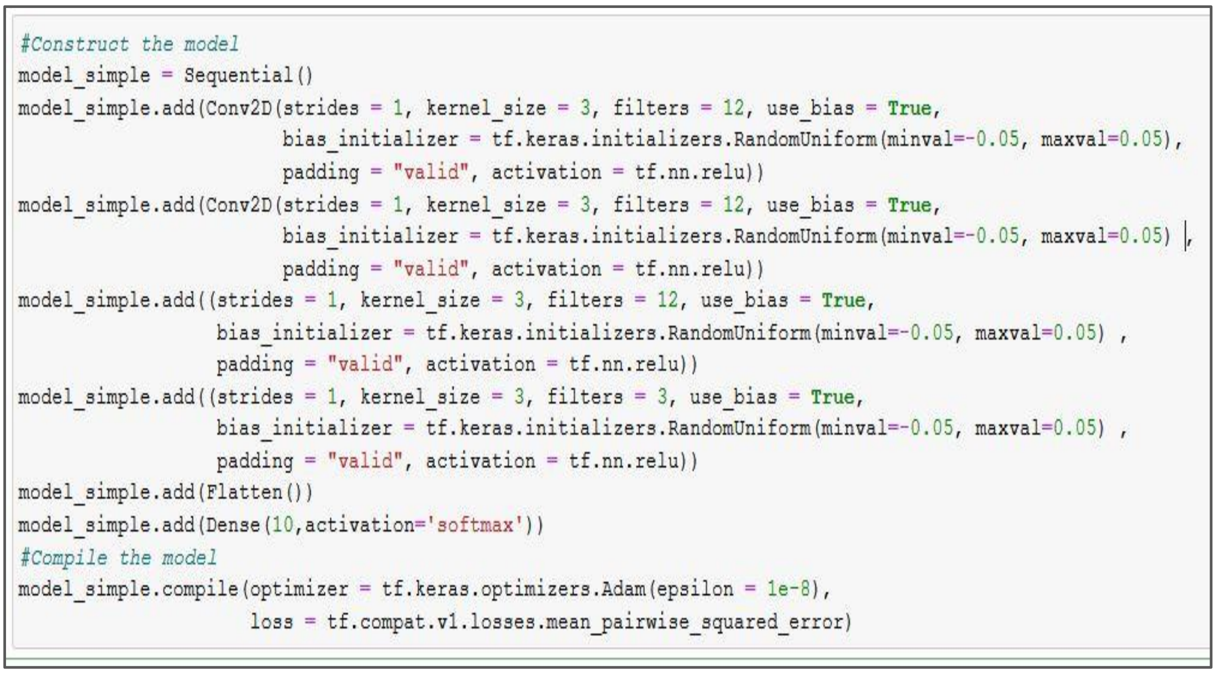
- Converting the input grayscale image into Lab color space
- Input the image having only the L channel (since it is a grayscale image) to the model
- Model predicts the corresponding a and b channel to the given L channel.

- Combining the given L channel with the predicted a b channel to give the final output, a colorized image.

With the evolution of Artificial Neural Network and color models, we were able to use our project to color the black and white images to colored images. The use of Convolutional Neural Network on image analysis made it possible for us to convert black and white images to colored.

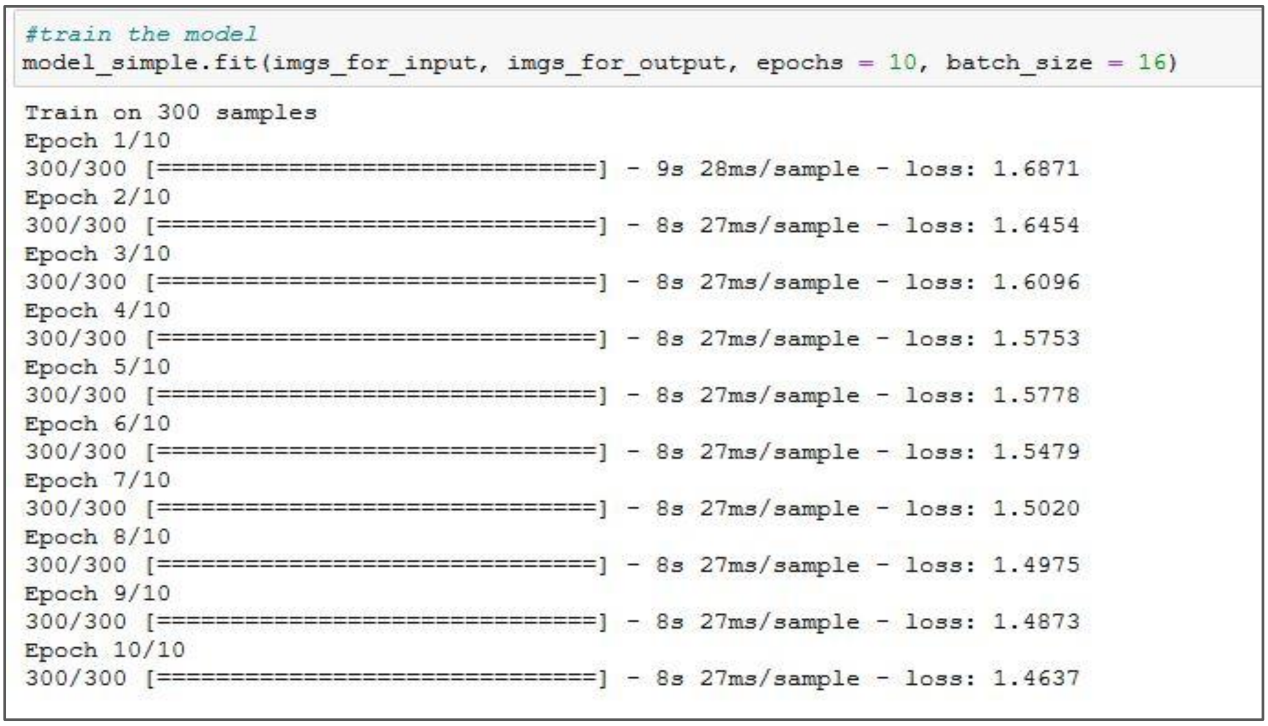
CNN Model

To execute the image colorization technique, we built a CNN model in the TensorFlow enabled Jupyter Notebook hosted in AWS Sagemaker. We first loaded the two numpy arrays of 'ab' and 'l' channels that we created during the data cleaning process. Then we create a model using Keras sequential model. We used it because it gives us the flexibility of adding the layers one by one in a sequential manner. Here, in our model due to memory and space constraints we could not add more layers to our model. Therefore, we used four convolutional layers with ReLu (Rectified Linear Unit) as the activation function, we used ReLu as our activation function because it has given the best results for image analysis. We used the *RandomUniform* as the initializer which generates tensors with uniform distribution i.e it generates random weights at the start of the network, but the random weights are normalized. We also used the padding value as *'valid'* it means that we kept the padding as zero (no padding at all), padding was not required because our analysis does not require to get the pixels of the corner of the images very precisely.



```
#Construct the model
model_simple = Sequential()
model_simple.add(Conv2D(strides = 1, kernel_size = 3, filters = 12, use_bias = True,
                        bias_initializer = tf.keras.initializers.RandomUniform(minval=-0.05, maxval=0.05),
                        padding = "valid", activation = tf.nn.relu))
model_simple.add(Conv2D(strides = 1, kernel_size = 3, filters = 12, use_bias = True,
                        bias_initializer = tf.keras.initializers.RandomUniform(minval=-0.05, maxval=0.05) ,
                        padding = "valid", activation = tf.nn.relu))
model_simple.add((strides = 1, kernel_size = 3, filters = 12, use_bias = True,
                  bias_initializer = tf.keras.initializers.RandomUniform(minval=-0.05, maxval=0.05) ,
                  padding = "valid", activation = tf.nn.relu))
model_simple.add((strides = 1, kernel_size = 3, filters = 3, use_bias = True,
                  bias_initializer = tf.keras.initializers.RandomUniform(minval=-0.05, maxval=0.05) ,
                  padding = "valid", activation = tf.nn.relu))
model_simple.add(Flatten())
model_simple.add(Dense(10,activation='softmax'))
#Compile the model
model_simple.compile(optimizer = tf.keras.optimizers.Adam(epsilon = 1e-8),
                     loss = tf.compat.v1.losses.mean_pairwise_squared_error)
```

Furthermore, to compile our model we used Adam optimizer as it has shown to give good results. The loss function used in the model is mean pairwise squared error, which instead of taking a difference of corresponding values of label and prediction, takes the difference between pairs of corresponding values of label and prediction. We then trained our model and used 10 epochs and the batch size of 16. The results of the trained model were not that good, and it was not able to colorize the image properly. So, we started researching about the CNN models that have already been trained on a large image dataset and give considerably good results.



```
#train the model
model_simple.fit(imgs_for_input, imgs_for_output, epochs = 10, batch_size = 16)
```

```
Train on 300 samples
Epoch 1/10
300/300 [==============================] - 9s 28ms/sample - loss: 1.6871
Epoch 2/10
300/300 [==============================] - 8s 27ms/sample - loss: 1.6454
Epoch 3/10
300/300 [==============================] - 8s 27ms/sample - loss: 1.6096
Epoch 4/10
300/300 [==============================] - 8s 27ms/sample - loss: 1.5753
Epoch 5/10
300/300 [==============================] - 8s 27ms/sample - loss: 1.5778
Epoch 6/10
300/300 [==============================] - 8s 27ms/sample - loss: 1.5479
Epoch 7/10
300/300 [==============================] - 8s 27ms/sample - loss: 1.5020
Epoch 8/10
300/300 [==============================] - 8s 27ms/sample - loss: 1.4975
Epoch 9/10
300/300 [==============================] - 8s 27ms/sample - loss: 1.4873
Epoch 10/10
300/300 [==============================] - 8s 27ms/sample - loss: 1.4637
```

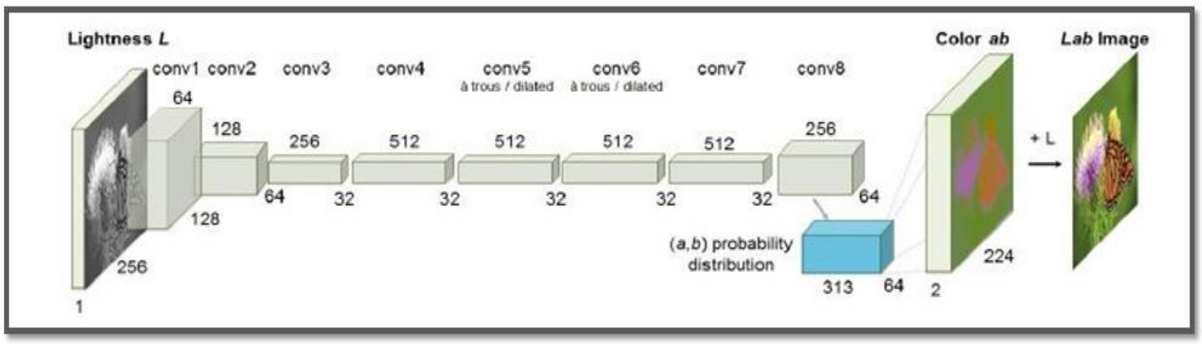
Caffe (Deep Learning Framework) and Model

CAFFE (Convolutional Architecture for Fast Feature Embedding) is a deep learning framework developed by researchers at UC Berkeley. It supports many types of deep learning architectures which mainly focus on image classification and image segmentation such as CNN, RCNN, LSTM and fully connected neural networks. It supports both GPU and CPU based acceleration computational **kernels libraries**.

We used the *'Colorful Image Colorization'* model (developed by Richard Zhang, et al.,) available on the Caffe Framework. This model is giving by far the best results for image colorization using convolutional Neural Networks. This model is already trained on 1 million images from the iMagenet dataset.
This model's architecture is described below:

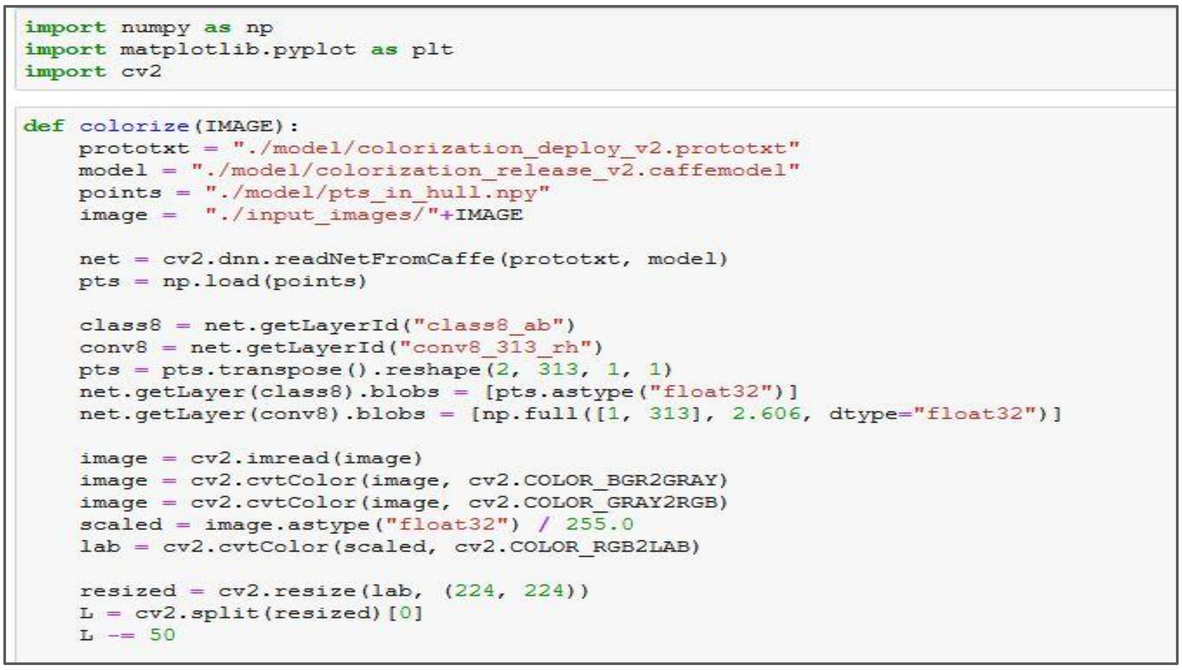
- The model has 8 convolution layers
- Each layer is comprised of conv and ReLU layers followed by BatchNorm layer
- Spatial downsampling and up sampling is used between each layer for changes in resolution
- The loss function used is tailored to colorization, it does class rebalancing by including a function that depends on the rarity of the colors
- The resultant colorized images from this model are more vibrant and perceptually realistic

Pictorial Representation of the Caffe model for image colorization



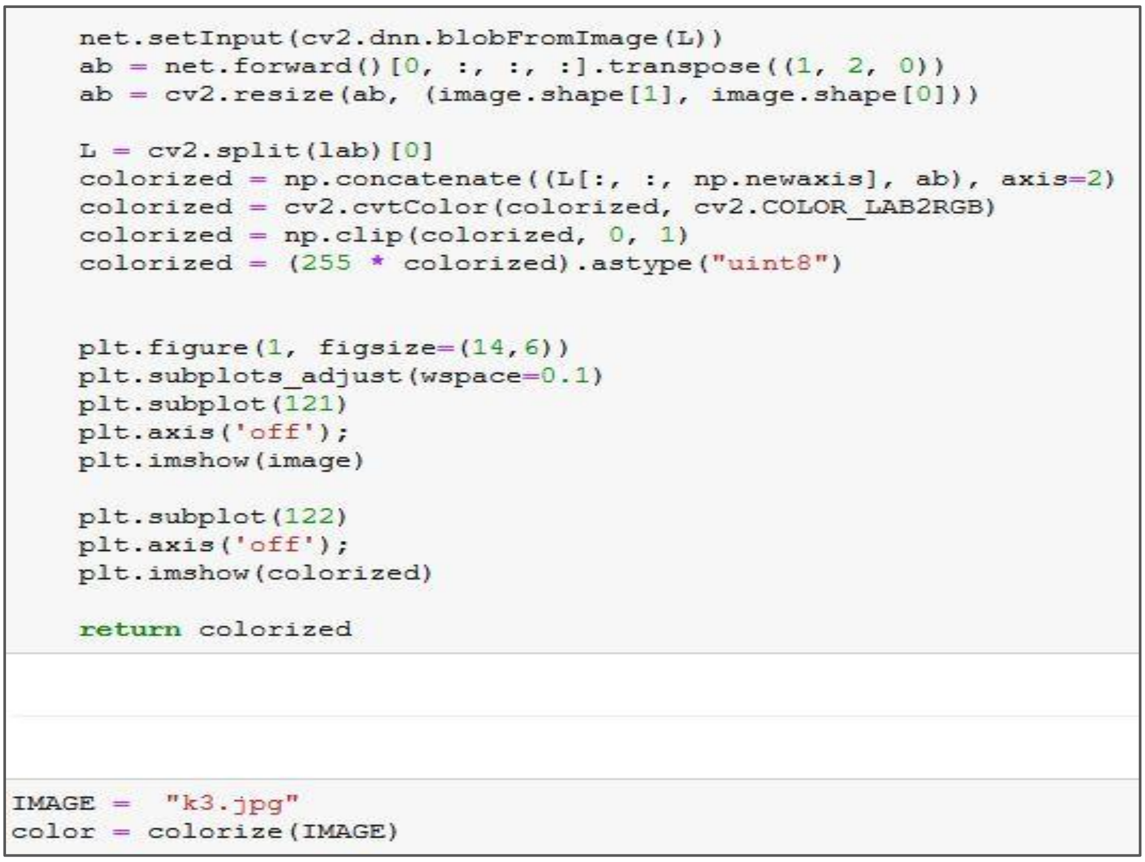
This model gets its realistic image colors on grayscale scale images by using a loss function which has a factor of class rebalancing in it. To elaborate, the dispensation of ab values in natural images is biased towards values having low ab values, due to the appearance of backgrounds such as clouds, pavement, dirt, and walls. We all might have observed that the pixels in natural images at desaturated values (colors such as blue, light green, brown i.e. low ab value) are higher than for saturated values (colors such as red, dark pink, orange, dark colors i.e. having high ab values). If we do not take this bias into account, then the loss function will be influenced by desaturated values of ab and will not take into account the influence of saturated ab values. The creators of this model tried to solve this problem by taking into account this class-imbalance problem by reweighing the loss of each pixel at the time of training the model based on the pixel color rarity. So, in this way this model is trying to remove the bias of the loss function. Thus, resulting in a better version of colorized image from a grayscale image or back/white image.

Screenshot of Caffe Model Code Execution



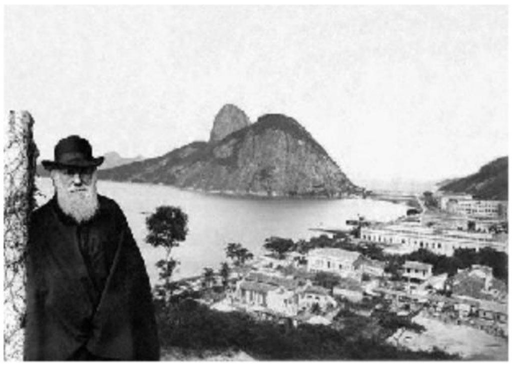
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

```
def colorize(IMAGE):
    prototxt = "./model/colorization_deploy_v2.prototxt"
    model = "./model/colorization_release_v2.caffemodel"
    points = "./model/pts_in_hull.npy"
    image =  "./input_images/"+IMAGE

    net = cv2.dnn.readNetFromCaffe(prototxt, model)
    pts = np.load(points)

    class8 = net.getLayerId("class8_ab")
    conv8 = net.getLayerId("conv8_313_rh")
    pts = pts.transpose().reshape(2, 313, 1, 1)
    net.getLayer(class8).blobs = [pts.astype("float32")]
    net.getLayer(conv8).blobs = [np.full([1, 313], 2.606, dtype="float32")]

    image = cv2.imread(image)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
    scaled = image.astype("float32") / 255.0
    lab = cv2.cvtColor(scaled, cv2.COLOR_RGB2LAB)

    resized = cv2.resize(lab, (224, 224))
    L = cv2.split(resized)[0]
    L -= 50
```

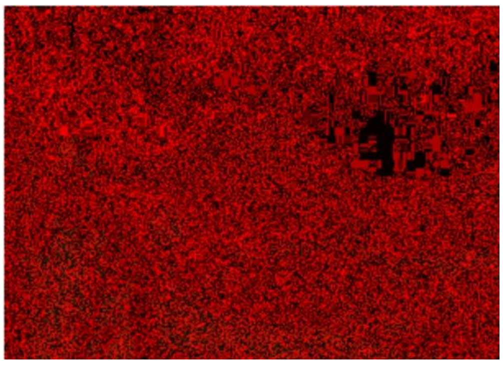


```
    net.setInput(cv2.dnn.blobFromImage(L))
    ab = net.forward()[0, :, :, :].transpose((1, 2, 0))
    ab = cv2.resize(ab, (image.shape[1], image.shape[0]))

    L = cv2.split(lab)[0]
    colorized = np.concatenate((L[:, :, np.newaxis], ab), axis=2)
    colorized = cv2.cvtColor(colorized, cv2.COLOR_LAB2RGB)
    colorized = np.clip(colorized, 0, 1)
    colorized = (255 * colorized).astype("uint8")


    plt.figure(1, figsize=(14,6))
    plt.subplots_adjust(wspace=0.1)
    plt.subplot(121)
    plt.axis('off');
    plt.imshow(image)

    plt.subplot(122)
    plt.axis('off');
    plt.imshow(colorized)

    return colorized
```

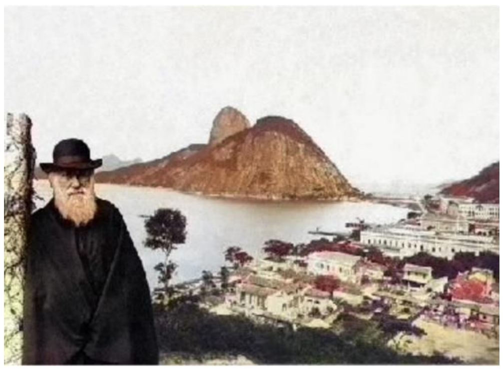
```
IMAGE =  "k3.jpg"
color = colorize(IMAGE)
```



Input Image



LAB Image



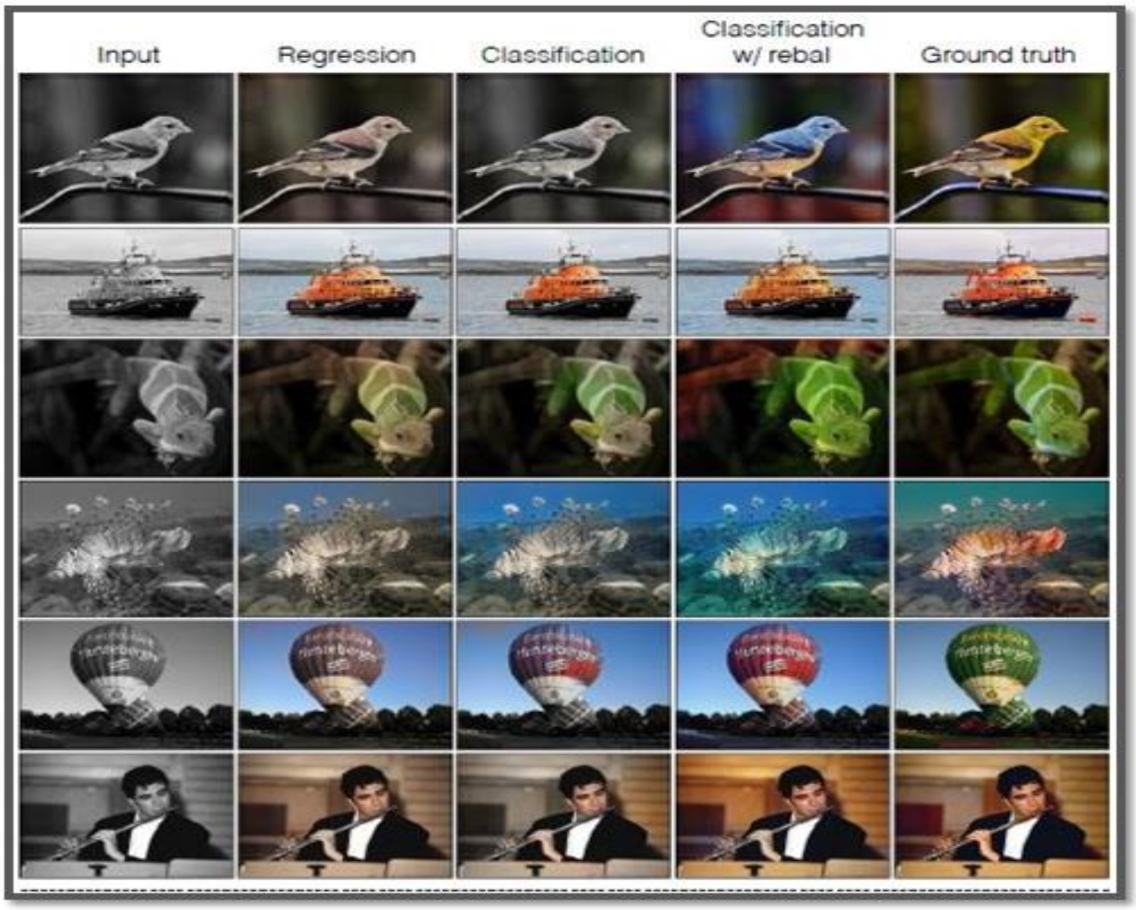
Colored Image

Data Visualization and Results

Data visualization is used to visualize the data and understand the patterns and take insights from it. Data visualization is an important part of an analytics project because as humans we tend to analyze and process better when information is given to us in visual form. Since our project is of image colorization, the colorized image is our data visualization and results at the same time. The colored image gives us a perspective of how well our model has worked to predict the colors corresponding to the given lightness intensity in the grayscale image. We can analyze and assess the power of the trained model by the output colorized image and then decide to reiterate the model and run it again. Hence our data visualization are the output images.

The researchers (Richard Zhang, et al.) have classified the colorized output image based on the model used. So according to their research and findings using a regression model will give the poorest results in terms of image colorization and will just increase the brightness of the same image, while classification (using CNN that we applied earlier in building our model from scratch) will give better results, but it will still be very different from the ground truth (the original colored image). But when this same classification model is used with class rebalancing loss function, it will give the best results i.e. the colorized images will look more similar to the ground truth.

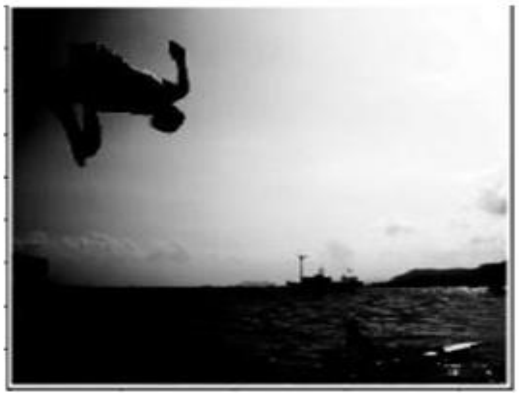
Expected Results from different Models

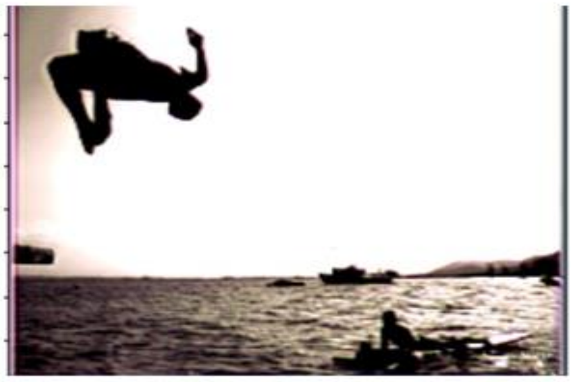


Below is the input and the colorized image from the model that we trained from scratch data and we have compared it with the colorized image of the Caffe model, which has given better colors to the image.

Model Trained from Scratch

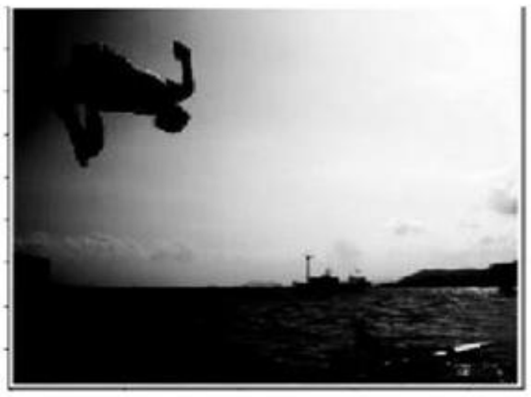
Input Image **Colorized Image**

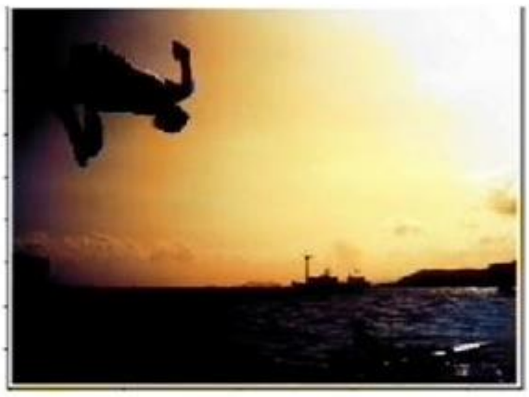




Caffe Model

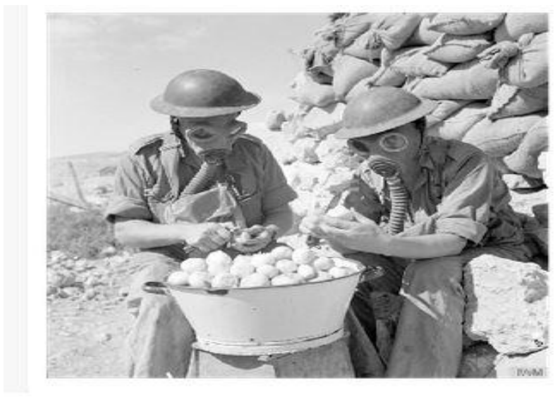
Input Image **Colorized Image**

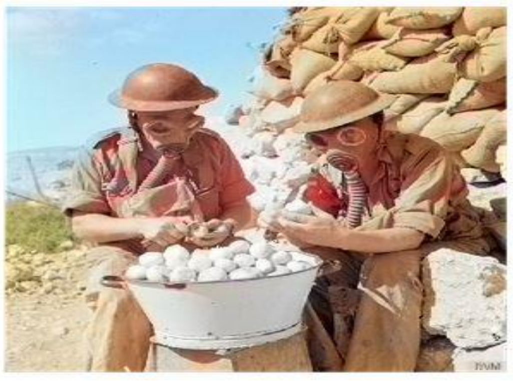




Caffe Model

Input Image **Colorized Image**





We can clearly visualize our results and assess that the Caffe model which is already pre trained on 1 Million Imagenet dataset gives far much better and accurate colorized images from the input grayscale image.

Challenges

While working on this project we hit against few challenges and we were able to resolve most of them. Some of the prominent challenges are list as below -

- While scraping data from Instagram and other sources, the selenium driver extracts the images displayed on the first page. For more images, the page must be scrolled for which we had to write a custom script.
- Understanding of complex technique behind the Deep Learning Neural Network, CNN
- Learning the working behind few AWS services in order to implement real time data scraping and modelling for our project
- As this project utilized the AWS free tier subscription, having more than ml.t2.medium instances was a challenge. If one needs to increase the instance size a request has to be raised to the support team and it takes 2-3 days to get a new instance.
- This project was running on a 4GB ram instance, which was the available free instance in the sagemaker, due to which we could not train on more than 4gb data, if given a better instance the results from the model would be of much accurate.

Learnings

While working on this project we got several opportunities to learn various new technologies particularly around Machine Learning and Cloud Platform. Most of the aspects included in this project were very new to us and we did not really have practical knowledge before. This project involves learnings as mentioned below -

- Researching and understanding of business implications involved in our project
- Learned about various AWS services and how to utilize cloud for all the activities involved in the project
- Data scraping using Selenium and Beautiful Soup
- Understanding Deep Learning Neural Networks (CNN) involved in image recognition and image colorization

Future Aspects

We have successfully worked on creating the project involving CNN model on cloud environment. We were able to successfully achieve a well-trained model which can provide the solution to our business problem. However, this project has huge scope in terms of the business applications as already mentioned and thus we have few plans to work on over and above our said deliverables. We will try our best to implement the below future aspects in future -

- Web application - We plan to build this solution as a web application which can take input as black and white image and can provide output to the users as a colored version of input. This would be helpful to the users in terms of reliving their old memories
- Video Coloring - This can be one of the helpful extensions to our project as we can also implement the same model to video's frames and produce the colored video as output after collaborating the colored frames as video
- Conditional GANs - This would be great learning opportunity if we would be able to implement GAN to produce the colored version of black white images and then comparing it with the output of CNN
- Improvising Colored Images - This would again be an interesting thing to learn and implement if we would also be able to sharpen and improvise the colored images

Conclusion

For this project we worked on colorizing the black and white images. The reason behind choosing this project was majorly dependent on its business implementation and utilizing deep learning models. We worked on implementing the project in a cloud environment.

As part of this project we researched on image processing and colorization to gather better understanding of these concepts. Understanding the business requirements and its implementation. We worked on deep analyzing few of the applications which would be benefitted by this project. For the purpose of this project we required a huge amount of image dataset and for that we did data sourcing and scraping from multiple websites. In order to achieve scraping through automation we used selenium and beautiful soup packages and created the function for the same using Jupyter notebook. We utilized AWS services for data storage, transformation and monitoring of logs. We worked on data cleaning, scaling pixel intensity of images and resizing of the LAB images to fit the colorization network and implemented the CNN and Caffe Model. We utilized AWS Sagemaker and Jupyter Network for modelling.

As per the recent hype around the images and videos, we hope to implement future aspects and work on improvising the project to one level up. We also plan to better our training model by training it on more data so that we can achieve even better results. We plan to build an automated product for image colorization and utilize AWS cloud services to scale and deploy our product.

References

1. https://mc.ai/coloring-black-white-images-using-deep-learning/
2. https://docs.aws.amazon.com/
3. https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
4. https://www.learnopencv.com/convolutional-neural-network-based-image-colorization-using-opencv/
5. https://towardsdatascience.com/colorizing-old-b-w-photos-and-videos-with-the-help-of-ai-76ba086f15ec
6. https://en.wikipedia.org/wiki/CIELAB_color_space
7. https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb
8. https://towardsdatascience.com/cnn-vs-fully-connected-network-for-image-processing-8c5b95d4e42f
9. https://theappsolutions.com/blog/development/convolutional-neural-networks/
10. https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7
11. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
12. https://www.allerin.com/blog/top-5-sources-of-big-data
13. http://cs231n.stanford.edu/reports/2017/pdfs/302.pdf
14. https://en.wikipedia.org/wiki/Caffe_(software)
15. https://www.tensorflow.org/guide/keras
16. https://stats.stackexchange.com/questions/383727/whats-the-difference-between-a-dense-layer-and-an-output-layer-in-a-cnn
17. http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/
18. https://cv-tricks.com/opencv/deep-learning-image-colorization/
19. https://caffe.berkeleyvision.org/gathered/examples/finetune_flickr_style.html
20. https://caffe2.ai/docs/tutorial-image-pre-processing.html
21. https://aws.amazon.com/blogs/machine-learning/call-an-amazon-sagemaker-model-endpoint-using-amazon-api-gateway-and-aws-lambda/
22. https://aws.amazon.com/blogs/machine-learning/build-test-and-deploy-your-amazon-sagemaker-inference-models-to-aws-lambda/
23. https://medium.com/@srujana.rao2/scraping-instagram-with-python-using-selenium-and-beautiful-soup-8b72c186a058
24. https://www.shellvoide.com/python/scraping-and-download-all-images-from-a-web-page-python/
25. https://towardsdatascience.com/web-scraping-using-selenium-python-8a60f4cf40ab