



INDIANA UNIVERSITY
BLOOMINGTON

ENGR-E516 Engineering Cloud Computing

FINAL PROJECT REPORT

SKIN LESION CLASSIFICATION FOR MELANOMA DETECTION

Sakshi Rathi
Sreshta Reddy N
Shruti Houji

December 8th, 2023

Prof. Dingwen Tao

1. Introduction:

Skin cancer, one of the most common and potentially life-threatening diseases globally, necessitates early detection for effective treatment and favorable patient outcomes. In today's technologically advanced era, the integration of cloud services has the potential to significantly enhance the efficiency and accuracy of skin cancer detection methodologies. This project is dedicated to the development of an innovative Skin Cancer Detection system, utilizing the robust capabilities of AWS services.

The primary aim is to create an end-to-end solution that employs machine learning and AWS cloud infrastructure to streamline the analysis of dermatoscopic images. This endeavor is not just a technological advancement but a beacon of hope for early and accurate skin cancer diagnosis, expediting necessary medical interventions.

The project predominantly targets medical researchers and analysts, marking a step towards integrating advanced cloud technology into medical research. Utilizing the comprehensive and diverse HAM10000 dataset, hosted by Harvard Dataverse, this project will process dermatoscopic images from various demographics. The dataset comprises 10015 images, encompassing a wide range of diagnostic categories, such as Actinic keratoses, Basal cell carcinoma, Benign keratosis-like lesions, Dermatofibroma, Melanoma, Melanocytic nevi, and Vascular lesions. Each category provides a unique challenge and opportunity for our machine learning models to learn and accurately predict.

Through this project, we aim to harness the power of cloud computing and machine learning to provide a reliable, scalable, and accessible tool for skin cancer detection. This tool is not just a technological marvel but a critical aid in the fight against skin cancer, potentially saving lives through early detection and diagnosis.

The primary objective is to develop a user-friendly and accurate skin cancer detection tool using machine learning algorithms and AWS cloud services. This tool will analyze dermatoscopic images for signs of malignancy and provide quick, reliable diagnoses. Our mission is to bridge the gap between advanced technology and medical research, offering a cutting-edge solution to a pressing global health issue.

2. Background and Related Work:

Background:

Skin cancer, one of the most prevalent forms of cancer globally, poses a significant health challenge. Traditional diagnosis methods, primarily based on dermatologists' visual examination, are not only subjective but also limited by the availability of specialists. This scenario emphasizes the need for more reliable, scalable, and rapid diagnostic methods.

In recent years, the field of dermatology has seen a paradigm shift with the introduction of Artificial Intelligence (AI) and Machine Learning (ML) in diagnosing skin cancers. Research in this area has primarily focused on using Convolutional Neural Networks (CNNs) and deep learning algorithms to analyze dermoscopic images. These advanced techniques offer the promise of higher accuracy, consistency, and the ability to handle large volumes of data compared to traditional methods. However, one of the significant challenges in this field is the deployment and management of these complex models, particularly in terms of resources and accessibility.

This project aims to advance the capabilities in this field by leveraging AWS cloud services to develop a robust and scalable skin cancer detection system. Utilizing AWS technologies like Amazon Rekognition and Amazon SageMaker, the project proposes a solution that not only enhances diagnostic accuracy but also addresses challenges related to scalability, cost, and global accessibility.

Related Work:

The integration of ML in skin cancer diagnosis has seen various research initiatives, focusing on different aspects of the problem. Key studies that have significantly contributed to this domain include:

1. “Deep Semantic Segmentation and Multi-Class Skin Lesion Classification Based on Convolutional Neural Network” (Anjum et al., 2020): This paper addresses the challenge of early-stage skin lesion diagnosis by proposing a three-phase model using CNNs. It demonstrates effective localization, segmentation, and classification of skin lesions, validating its approach on the MICCAI ISIC challenges datasets from 2017 to 2019.
2. “An approach for multiclass skin lesion classification based on ensemble learning” (Rahman et al., 2021): This study introduces a weighted ensemble model combining five deep neural networks to classify skin lesions. It achieved a recall score of 94% using the ISIC 2019 dataset, showcasing the effectiveness of ensemble methods in skin cancer classification.
3. “Skin Lesions Classification into Eight Classes Using Deep Convolutional Neural Network and Transfer Learning” (Kassem et al., 2019): Focusing on classification accuracy, this research utilized pre-trained models and transfer learning for classifying skin lesions into eight categories, demonstrating a high accuracy rate of 94.92%.
4. “WonDerM: Skin Lesion Classification with Fine-tuned Neural Networks” (Lee et al., 2018): This paper highlights the use of neural networks fine-tuned with segmentation data, achieving notable accuracy in classifying skin lesions into different types.
5. “Skin Lesion Classification Using Hybrid Deep Neural Networks” (Mahbod et al., 2019): The study compares CNNs with classical methods, illustrating the superiority of CNNs in melanoma classification with an accuracy of 83.83%.

These studies collectively underscore the potential of ML and AI in revolutionizing skin cancer diagnosis. They highlight the strengths of various approaches, from ensemble models and transfer

learning to deep CNNs. However, they also point out the existing gaps, particularly in model interpretability, explainability, and real-world deployment challenges.

In summary, this project builds upon these foundational studies, aiming to create a more accessible and scalable system using cloud technology. By doing so, it seeks to make advanced skin cancer detection technology more readily available, contributing significantly to early diagnosis and treatment interventions.

3. Proposed Design/System/Architecture/Application

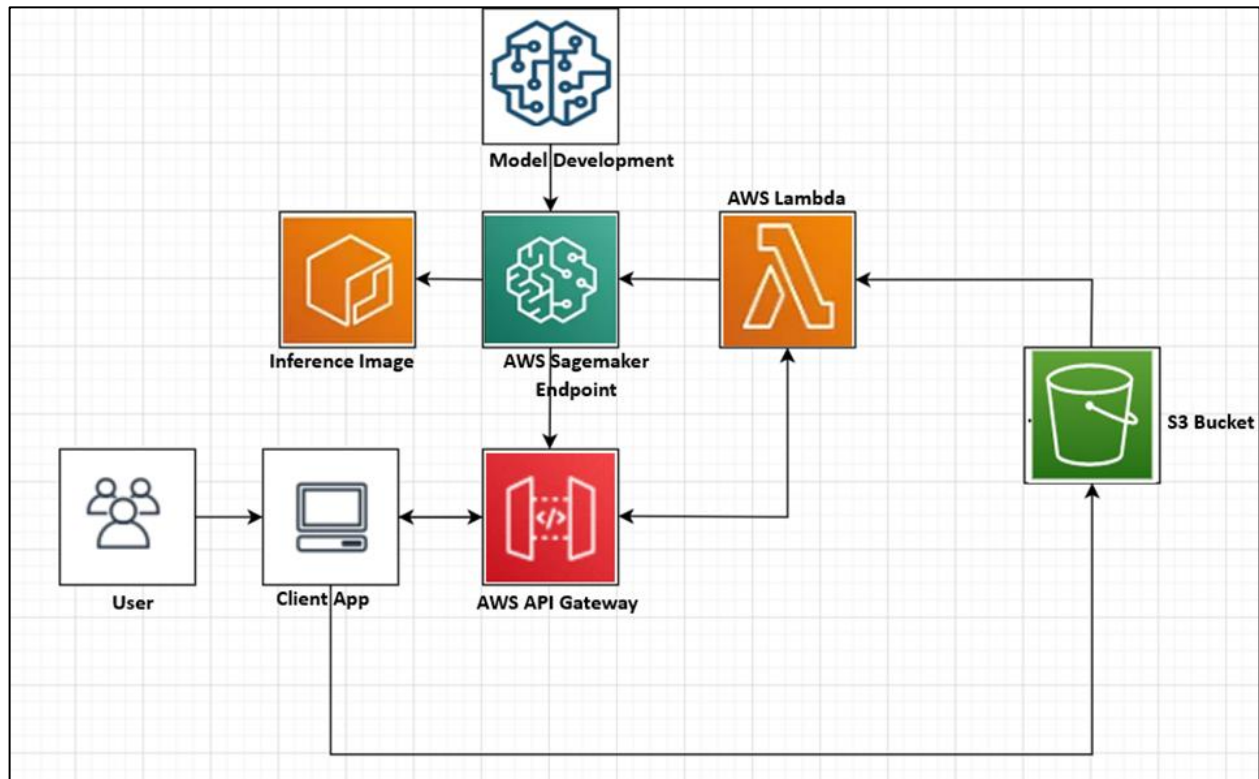


Figure 1 : Architecture Diagram for Skin Cancer Detection System Using AWS Services

AWS services used:

- AWS S3 bucket-** Amazon Simple Storage Service (Amazon S3) is an object storage service built to store and retrieve any amount of data from anywhere. Data is stored as objects within resources called “buckets”. Our project's dataset is stored in the S3 bucket named 'ecc-project-dataset', which serves as the primary repository for storing dermatoscopic images.

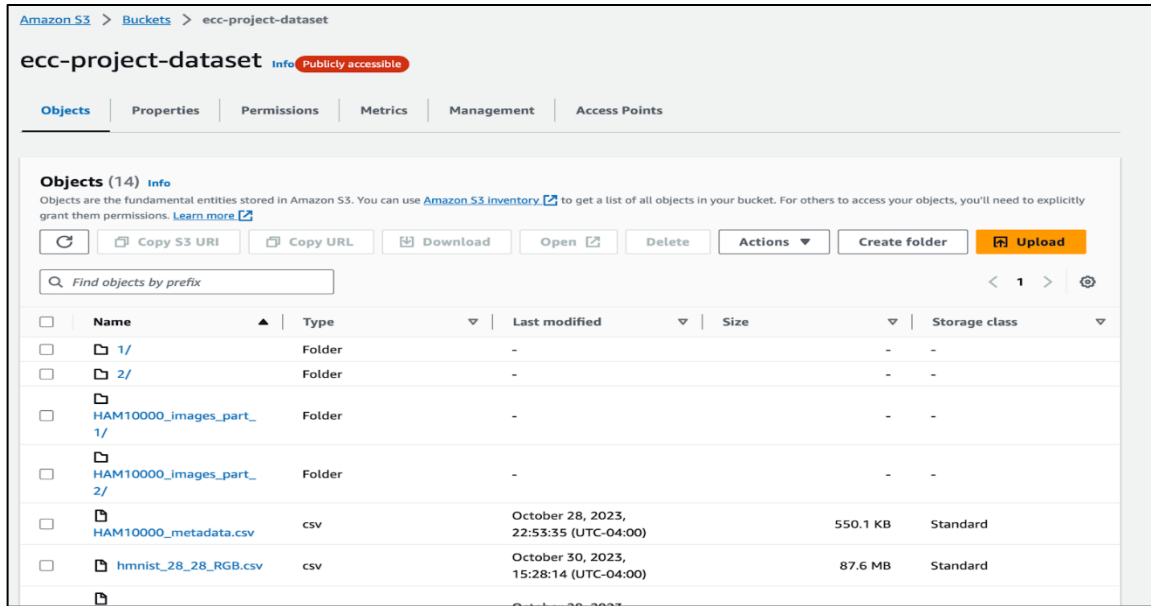


Figure 2: Illustrates the S3 bucket interface showing the stored dataset.

- b) **AWS Sagemaker notebook instance**- Employed for the entire machine learning pipeline, including preparing data, process data, train machine learning and deep learning model, deploy models, test and validate the model. This notebook instance is crucial for developing and testing our machine learning models.

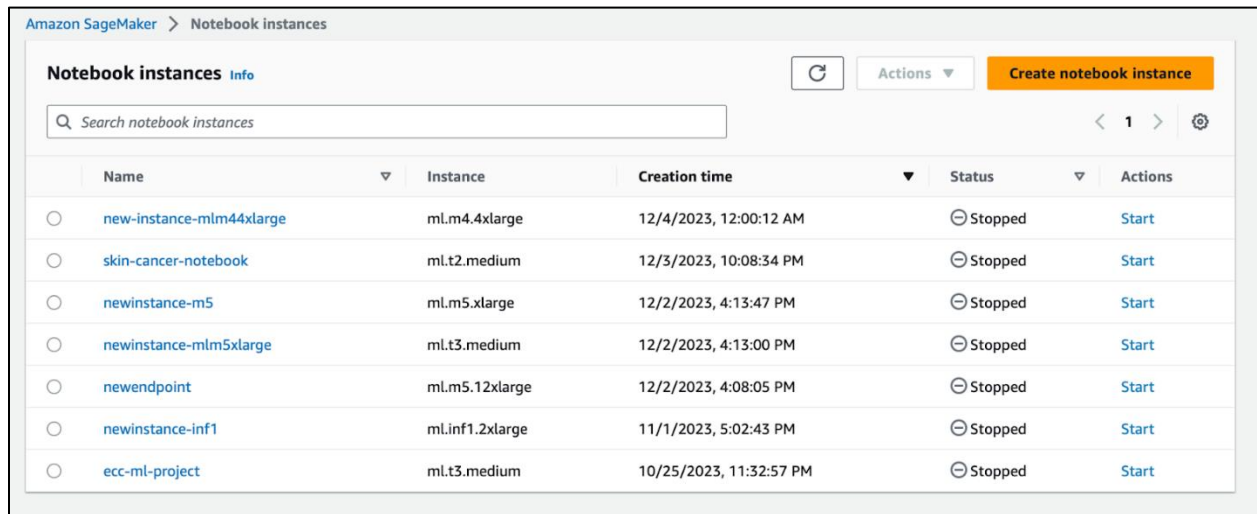


Figure 3: Depicts the SageMaker notebook instance used for model development

- c) **AWS service endpoint**- We have two endpoints created for xgboost model and monai pytorch model typically refers to the URL where a deployed model is accessible for making predictions or inferences. These endpoints are invoked in the lambda function in AWS lambda to train the backend model for web application without using a server.

| Amazon SageMaker > Endpoints | | | | | |
|--|--|--|------------------------|--------------------------|------------------------|
| Endpoints | | | | | |
| <input type="text" value="Search endpoints"/> < 1 > | | | | | |
| | Name | ARN | Creation time | Status | Last updated |
| <input type="radio"/> | xgboost-2023-12-06-15-37-49-960 | arn:aws:sagemaker:us-east-2:087733582531:endpoint/xgboost-2023-12-06-15-37-49-960 | 12/6/2023, 10:37:51 AM | ✔ InService | 12/6/2023, 10:40:04 AM |
| <input type="radio"/> | pytorch-training-2023-12-04-16-41-34-400 | arn:aws:sagemaker:us-east-2:087733582531:endpoint/pytorch-training-2023-12-04-16-41-34-400 | 12/4/2023, 11:41:38 AM | ✔ InService | 12/4/2023, 11:44:03 AM |

Figure 4: Shows the configuration of AWS service endpoints for model access

- d. **AWS IAM** - It is used to define roles and its associated permission policies for that particular role. Below are the set of policies we have defined for the role AmazonSageMaker-Execution and AmazonSageMakerServiceCatalogProductsLambdaRole which can help to access the resources to other services of amazon.

Displaying the IAM roles and policy configurations for SageMaker and Lambda functions

AmazonSageMaker-ExecutionRole-20231025T233248 [Info](#) Delete

SageMaker execution role created from the SageMaker AWS Management Console.

Summary Edit

Creation date
October 25, 2023, 23:32 (UTC-04:00)

Last activity
✔ 24 minutes ago

ARN
[arn:aws:iam::087733582531:role/service-role/AmazonSageMaker-ExecutionRole-20231025T233248](#)

Maximum session duration
1 hour

Permissions Trust relationships Tags Access Advisor Revoke sessions

Permissions policies (11) [Info](#)
Refresh Simulate Remove Add permissions

You can attach up to 10 managed policies.

Filter by Type: All types
< 1 >

| <input type="checkbox"/> | Policy name | Type | Attached entities |
|--------------------------|--|-------------|-------------------|
| <input type="checkbox"/> | AmazonS3FullAccess | AWS managed | 6 |
| <input type="checkbox"/> | AmazonS3ReadOnlyAccess | AWS managed | 1 |

Figure 5 : AmazonSageMaker-Execution Role

Permissions policies (11)
[Info](#)

Simulate

Remove

Add permissions

You can attach up to 10 managed policies.

Search

Filter by Type

All types

<

1

>

| <input type="checkbox"/> | Policy name | Type | Attached entities |
|--------------------------|--|------------------|-------------------|
| <input type="checkbox"/> | AmazonS3FullAccess | AWS managed | 6 |
| <input type="checkbox"/> | AmazonS3ReadOnlyAccess | AWS managed | 1 |
| <input type="checkbox"/> | AmazonSageMaker-ExecutionPolicy-202... | Customer managed | 1 |
| <input type="checkbox"/> | AmazonSageMakerFullAccess | AWS managed | 6 |
| <input type="checkbox"/> | AWSLambda_FullAccess | AWS managed | 1 |
| <input type="checkbox"/> | AWSLambda_ReadOnlyAccess | AWS managed | 1 |
| <input type="checkbox"/> | AWSLambdaBasicExecutionRole | AWS managed | 1 |
| <input type="checkbox"/> | AWSLambdaExecute | AWS managed | 1 |
| <input type="checkbox"/> | AWSLambdaRole | AWS managed | 1 |
| <input type="checkbox"/> | s3-getobject | Customer inline | 0 |
| <input type="checkbox"/> | s3-permission-lambda | Customer inline | 0 |

Figure 6 : Permissions for AmazonSageMaker-Execution

[IAM](#) > [Roles](#) > [AmazonSageMakerServiceCatalogProductsLambdaRole](#)

AmazonSageMakerServiceCatalogProductsLambdaRole
[Info](#)

Delete

SageMaker role created from the SageMaker AWS Management Console. This role will grant permissions required to use AWS Lambda within the Amazon SageMaker portfolio of products.

Summary

Edit

Creation date

October 30, 2023, 14:33 (UTC-04:00)

ARN

arn:aws:iam::087733582531:role/service-role/AmazonSageMakerServiceCatalogProductsLambdaRole

Last activity

Yesterday

Maximum session duration

1 hour

Permissions

Trust relationships

Tags

Access Advisor

Revoke sessions

Permissions policies (4)
[Info](#)

Simulate

Remove

Add permissions

You can attach up to 10 managed policies.

Search

Filter by Type

All types

<

1

>

| <input type="checkbox"/> | Policy name | Type | Attached entities |
|--------------------------|------------------------------------|-------------|-------------------|
| <input type="checkbox"/> | AmazonS3FullAccess | AWS managed | 6 |

Figure 7 : AmazonSageMakerServiceCatalogProductsLambdaRole

| Permissions | | | | Trust relationships | Tags | Access Advisor | Revoke sessions |
|--|---|------------------|-------------------|--|------|----------------|-----------------|
| Permissions policies (4) Info Refresh Simulate Remove Add permissions | | | | | | | |
| You can attach up to 10 managed policies. | | | | | | | |
| <input type="text" value="Search"/> | | | | Filter by Type: All types | | | |
| <input type="checkbox"/> | Policy name | Type | Attached entities | | | | |
| <input type="checkbox"/> | AmazonS3FullAccess | AWS managed | 6 | | | | |
| <input type="checkbox"/> | AmazonSageMakerFullAccess | AWS managed | 6 | | | | |
| <input type="checkbox"/> | AmazonSageMakerServiceCatalogPro... | AWS managed | 1 | | | | |
| <input type="checkbox"/> | getObject | Customer managed | 1 | | | | |

Figure 8 : Permissions for AmazonSageMakerServiceCatalogProductsLambdaRole

e. **AWS Lambda** - AWS lambda is used to access an endpoint by integrating by directly invoking the Lambda function through an API Gateway. We have created two lambda functions:

- 1) lambda-function - we have defined this AWS Lambda function for xgboost endpoint that extracts and sends JSON data to a SageMaker endpoint for inference. As we have trained the xgboost model on the pixel values of an image, so it assumes a text/csv file content type and returns an integer result.

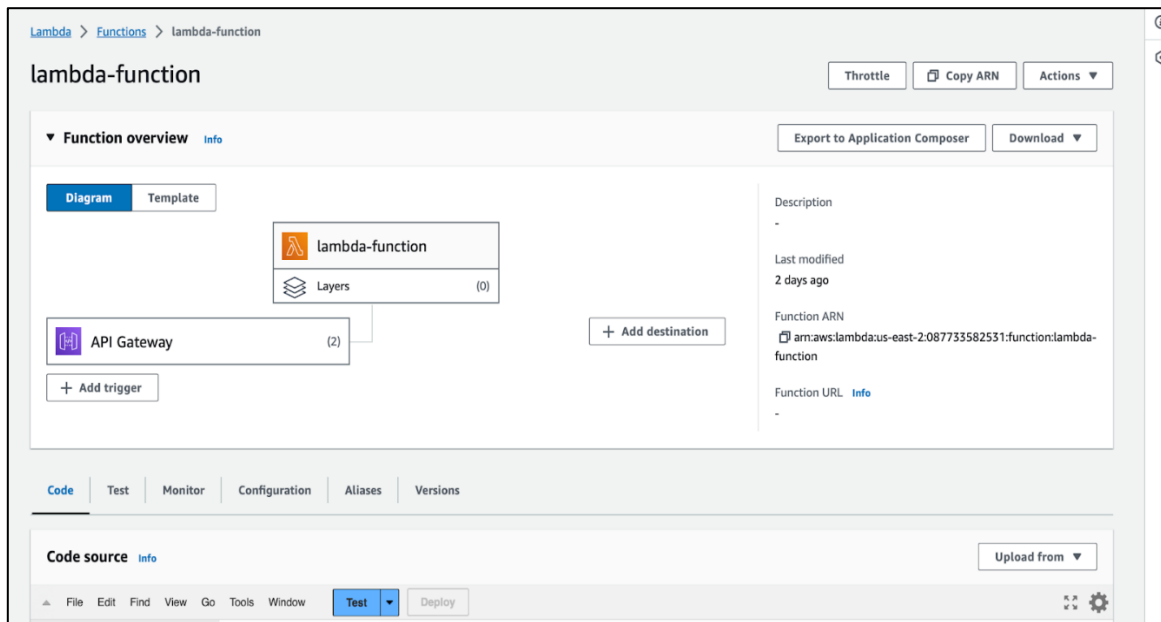


Figure 9 : Showcasing the Lambda function setup for the XGBoost model.


```

51
52 import os
53 import io
54 import boto3
55 import json
56 import csv
57
58 # grab environment variables
59 ENDPOINT_NAME = os.environ['ENDPOINT_NAME']
60 runtime= boto3.client('runtime.sagemaker')
61
62 def lambda_handler(event, context):
63     print("Received event: " + json.dumps(event, indent=2))
64
65     data = json.loads(json.dumps(event))
66     payload = data['data']
67     print(payload)
68
69     response = runtime.invoke_endpoint(EndpointName=ENDPOINT_NAME,
70                                     ContentType='text/csv',
71                                     Body=payload)
72
73     print(response)
74     result = json.loads(response['Body'].read().decode())
75     r=int(result)
76     print("this is a result:",r)
77
78     return r
79

```

Figure. 10 : Lambda function for the XGBoost model.

- 2) MyImageProcessingFunction - we have defined this AWS Lambda function for MONAI PyTorch model which downloads an image from an S3 bucket based obtained from the frontend through a react web application on the provided 'imageKey' in the query string parameters. It preprocesses the image, converting it to a NumPy array and normalizing pixel values. The processed image is then sent as JSON to a SageMaker endpoint and the result from the inference is returned.

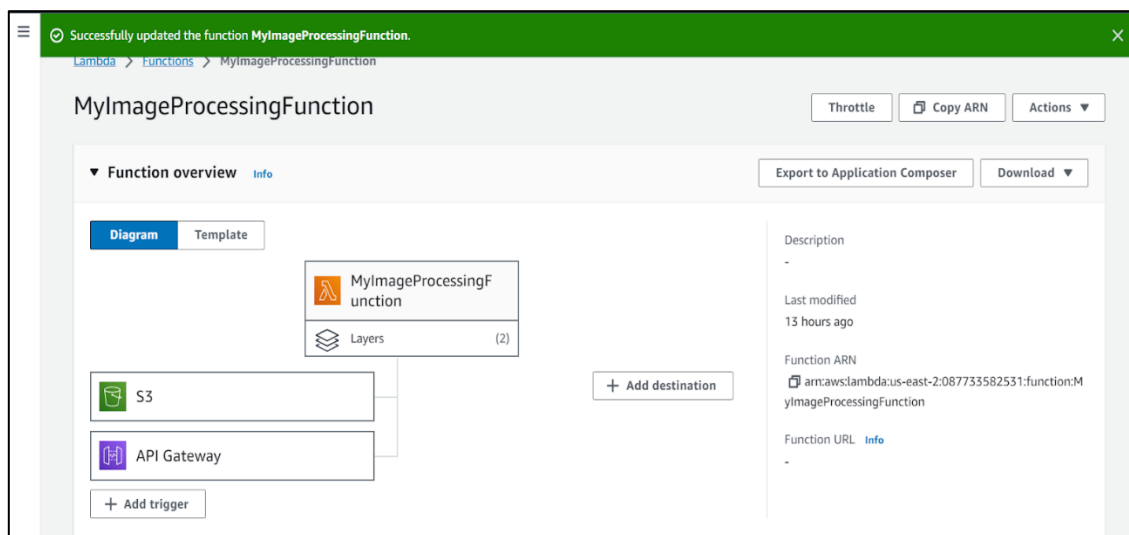


Figure. 11 : Lambda function setup for image preprocessing and PyTorch model inference.

```

8 s3_client = boto3.client('s3')
9 sagemaker_runtime = boto3.client('sagemaker-runtime')
10
11 def preprocess_image(image_bytes):
12     # Function to replicate the transformations applied in the SageMaker notebook
13     # Adjust this function based on the actual transformations used in val_transforms
14     image = Image.open(io.BytesIO(image_bytes))
15     image = image.resize((image_width, image_height)) # Resize the image
16     image_np = np.array(image) / 255.0 # Normalize the pixel values
17     if len(image_np.shape) == 3 and image_np.shape[2] == 3: # If the image has 3 channels
18         image_np = np.transpose(image_np, (2, 0, 1)) # Change to channel-first format
19     return image_np
20
21 def lambda_handler(event, context):
22     image_key = event.get('queryStringParameters', {}).get('imageKey')
23     if not image_key:
24         return {
25             'statusCode': 400,
26             'body': json.dumps("Missing or incorrect queryStringParameters")
27         }
28
29     bucket_name = 'ecc-project-dataset'
30
31     try:
32         # Download the image from S3
33         response = s3_client.get_object(Bucket=bucket_name, Key=image_key)
34         image_bytes = response['Body'].read()
35
36         # Preprocess the image
37         image_np = preprocess_image(image_bytes)
38         payload = json.dumps(image_np.tolist())

```

```

    # Invoke the SageMaker endpoint
    endpoint_response = sagemaker_runtime.invoke_endpoint(
        EndpointName='pytorch-training-2023-12-04-16-41-34-400',
        ContentType='application/json',
        Body=payload
    )

    result = json.loads(endpoint_response['Body'].read().decode())
    return {
        'statusCode': 200,
        'headers': {
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps(result)
    }

except Exception as e:
    return {
        'statusCode': 500,
        'headers': {
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps(f"Error processing request: {str(e)}")
    }

```

Figure. 12 : Lambda function for image preprocessing and PyTorch model inference.

- f. **AWS API Gateway** - We have created two REST APIs one for the xgboost model and other for the pytorch model. Manages REST APIs for both the XGBoost and PyTorch models, enabling the web application's frontend to interact with the backend models efficiently.

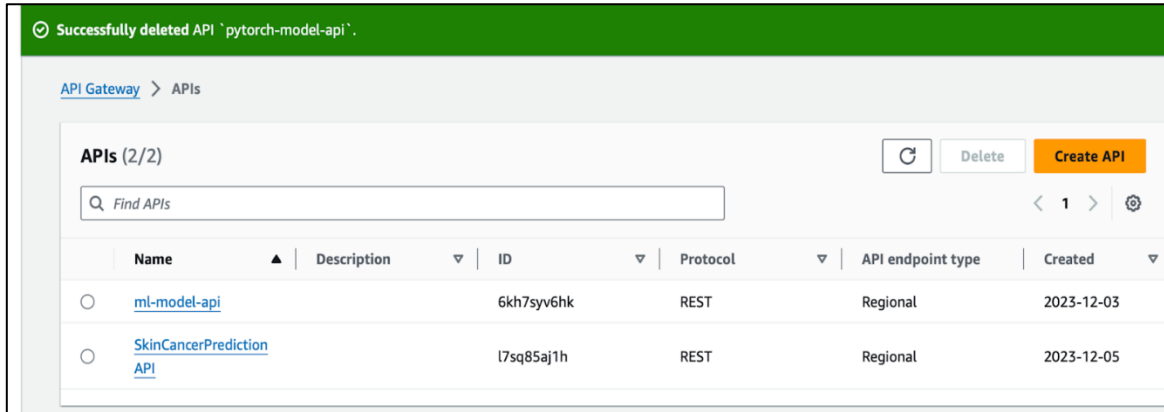


Figure. 13 : Outlines the API Gateway setup for model integration.

1. **ml-model-api** : We have created this REST API by creating a POST method which gets the input of pixel values of an image from the user and predicts the class for the type of cancer.

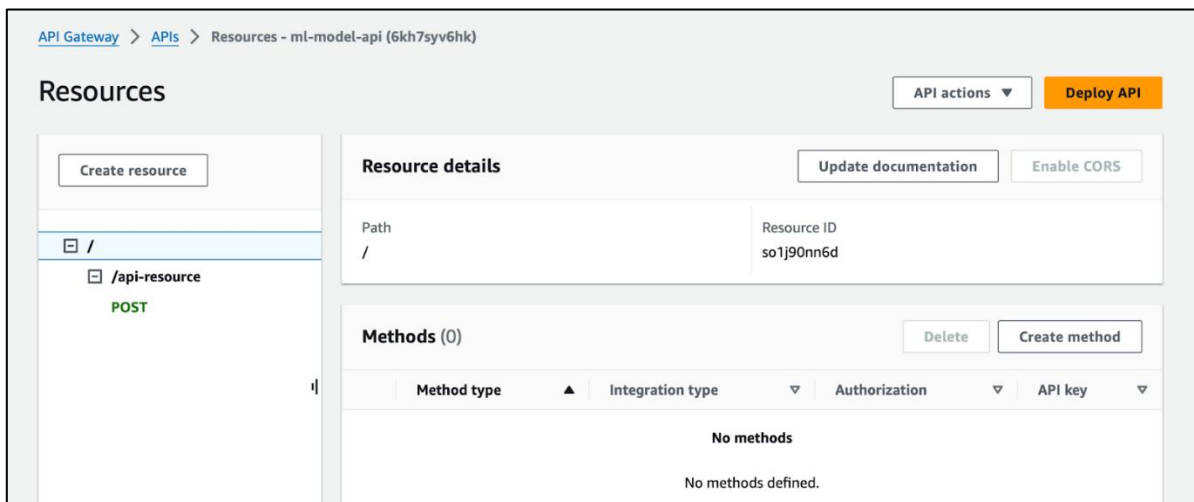


Figure. 14 : Demonstrates the API setup for the XGBoost model.

2. **SkinCancerPredcitionAPI** : We have created this REST API by creating a GET method for the pytorch model which gets the preprocessed input image integrated with the lambda function by triggering an endpoint to predict a class on the UI created for the application using React.

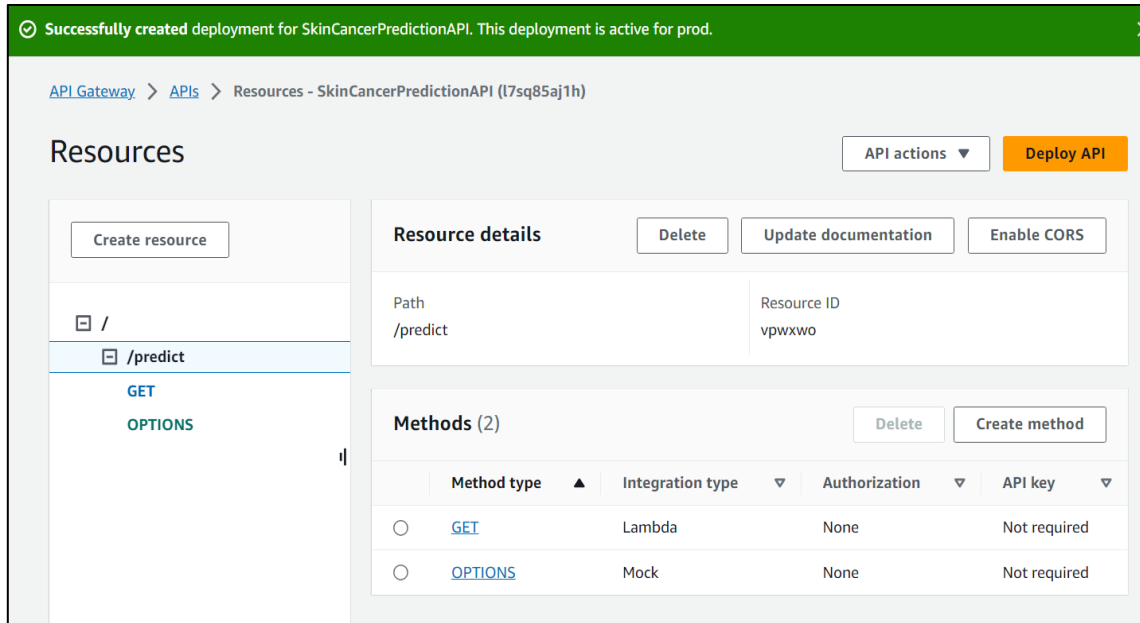


Figure 15 : Shows the API setup for the PyTorch model.

Summary of system's architecture and workflow for the project:

1. Image Upload and Storage

Users upload images via the website. These images are automatically stored in an AWS S3 bucket, a secure and scalable cloud storage service from AWS.

2. Lambda Function Trigger

Uploading an image to the S3 bucket automatically triggers an AWS Lambda function. AWS Lambda is a serverless computing service that executes code in response to events like image uploads, operating without the need for server management.

3. Model Prediction with SageMaker

The Lambda function, once triggered, interacts with a machine learning model hosted on AWS SageMaker. AWS SageMaker is a comprehensive service for building, training, and deploying machine learning models. The model hosted on SageMaker is trained to classify various skin diseases.

4. Analysis and Prediction

The SageMaker model processes the received image to predict the classification of the skin disease. It evaluates the image and calculates the probabilities of the skin condition belonging to specific disease classes.

5. API Gateway for REST API Responses

API Gateway, another component of the system, handles REST API calls. While it does not directly participate in the model inference process, it plays a crucial role in managing other API interactions on the website.

6. Displaying Results to the User

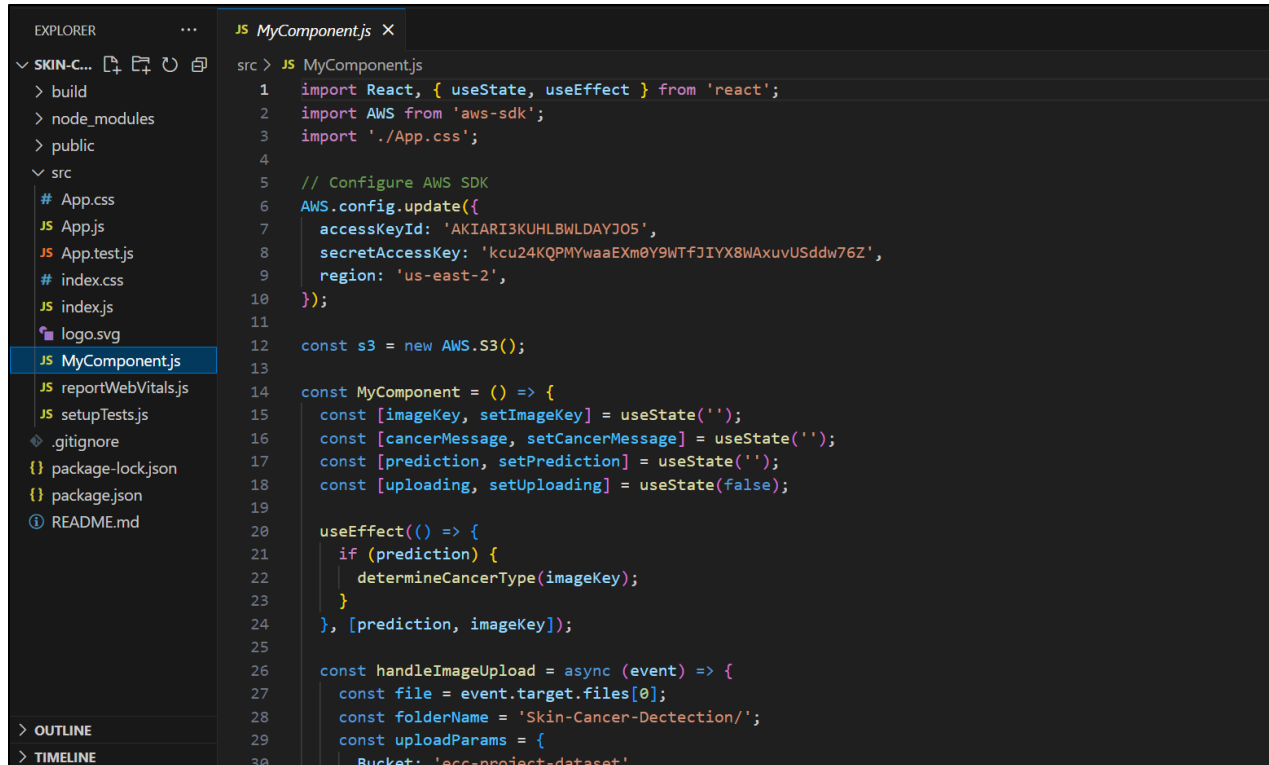
The prediction results from the SageMaker model are relayed back through the Lambda function to the website. The website then presents these results to the user, offering insights into the potential type of skin disease depicted in the image.

4. User Interface/ Front-End:

The user interface (UI) of our Skin Cancer Prediction system, developed using React.js, offers a straightforward and interactive platform for users to upload dermatoscopic images for analysis. The UI is designed to ensure ease of use while maintaining a clean aesthetic that aligns with medical professionalism.

- **Image Upload :** Users are greeted with a simple and clear option to upload an image of their skin lesion. This functionality is highlighted by a prominent "Choose File" button, which, when clicked, opens the user's file system to select an image for upload.
- **Prediction Trigger:** After selecting the image, users can initiate the prediction process by clicking the "Predict Skin Cancer" button. This action triggers the front-end JavaScript to validate the uploaded image and send it to the backend AWS infrastructure for processing.
- **Processing Feedback:** Upon clicking the prediction button, the system provides immediate visual feedback to inform the user that their image is being processed. This is crucial for maintaining engagement and trust, as the prediction process might take a few seconds.
- **Results Display:** Once the image is processed and a prediction is made by the backend model, the results are displayed to the user on the same UI. The system is designed to present the results in an easily understandable format, potentially including the type of skin lesion and the confidence level of the model's prediction.
- **User-Centric Design:** The UI's design incorporates elements that are visually appealing and do not overwhelm the user with medical jargon. It aims to demystify the prediction process, making advanced skin cancer detection technology accessible to a broader audience.
- **Accessibility and Responsiveness:** The UI is developed to be responsive, ensuring that users on various devices, including mobile phones and tablets, have equal access to the system's capabilities.

In summary, the UI serves as the entry point to the Skin Cancer Detection system, embodying the principles of simplicity, user engagement, and educational value, while ensuring a seamless experience from image upload to result dissemination.



```
1 import React, { useState, useEffect } from 'react';
2 import AWS from 'aws-sdk';
3 import './App.css';
4
5 // Configure AWS SDK
6 AWS.config.update({
7   accessKeyId: 'AKIARI3KUHLBWLDAYJ05',
8   secretAccessKey: 'kcu24KQPMYwaaEXm0Y9WTFJlYX8WAxuvUSddw76Z',
9   region: 'us-east-2',
10 });
11
12 const s3 = new AWS.S3();
13
14 const MyComponent = () => {
15   const [imageKey, setImageKey] = useState('');
16   const [cancerMessage, setCancerMessage] = useState('');
17   const [prediction, setPrediction] = useState('');
18   const [uploading, setUploading] = useState(false);
19
20   useEffect(() => {
21     if (prediction) {
22       determineCancerType(imageKey);
23     }
24   }, [prediction, imageKey]);
25
26   const handleImageUpload = async (event) => {
27     const file = event.target.files[0];
28     const folderName = 'Skin-Cancer-Dectection/';
29     const uploadParams = {
30       Bucket: 'ecc-project-dataset',
```

Figure 16 : React JS Code for the UI

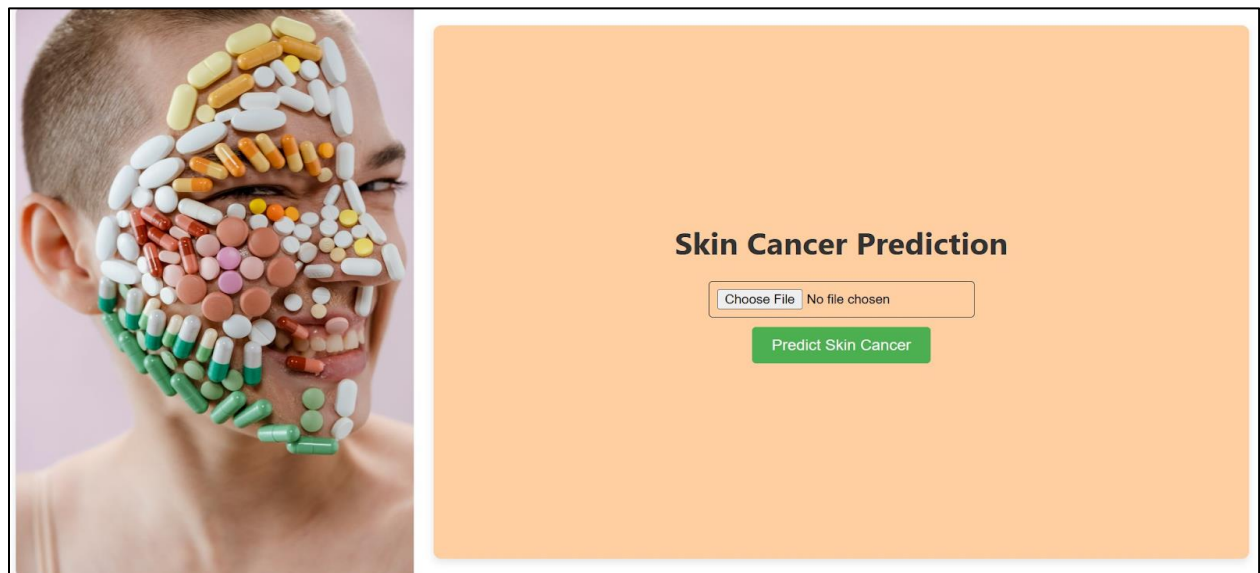


Figure 17 : UI Front-Page

Hosted the website on AWS S3 : <http://hostingskincancer.s3-website.us-east-2.amazonaws.com/>

5. Evaluation/Result Analysis:

Visualizations:

We have created one more column to our dataframe which has the path to the image where it is saved on the local machine of amazon sagemaker notebook instance:

Out[4]:

| | lesion_id | image_id | dx | dx_type | age | sex | localization | path |
|---|-------------|--------------|-----|---------|------|------|--------------|--|
| 0 | HAM_0000118 | ISIC_0027419 | bkl | histo | 80.0 | male | scalp | s3://ecc-project-dataset/HAM10000_images_part_1/ISIC_0027419.jpg |
| 1 | HAM_0000118 | ISIC_0025030 | bkl | histo | 80.0 | male | scalp | s3://ecc-project-dataset/HAM10000_images_part_1/ISIC_0025030.jpg |
| 2 | HAM_0002730 | ISIC_0026769 | bkl | histo | 80.0 | male | scalp | s3://ecc-project-dataset/HAM10000_images_part_1/ISIC_0026769.jpg |
| 3 | HAM_0002730 | ISIC_0025661 | bkl | histo | 80.0 | male | scalp | s3://ecc-project-dataset/HAM10000_images_part_1/ISIC_0025661.jpg |
| 4 | HAM_0001466 | ISIC_0031633 | bkl | histo | 75.0 | male | ear | s3://ecc-project-dataset/HAM10000_images_part_2/ISIC_0031633.jpg |

Figure. 18: Path to the image is added to the dataframe

1) Total count of images for each category of the cancer

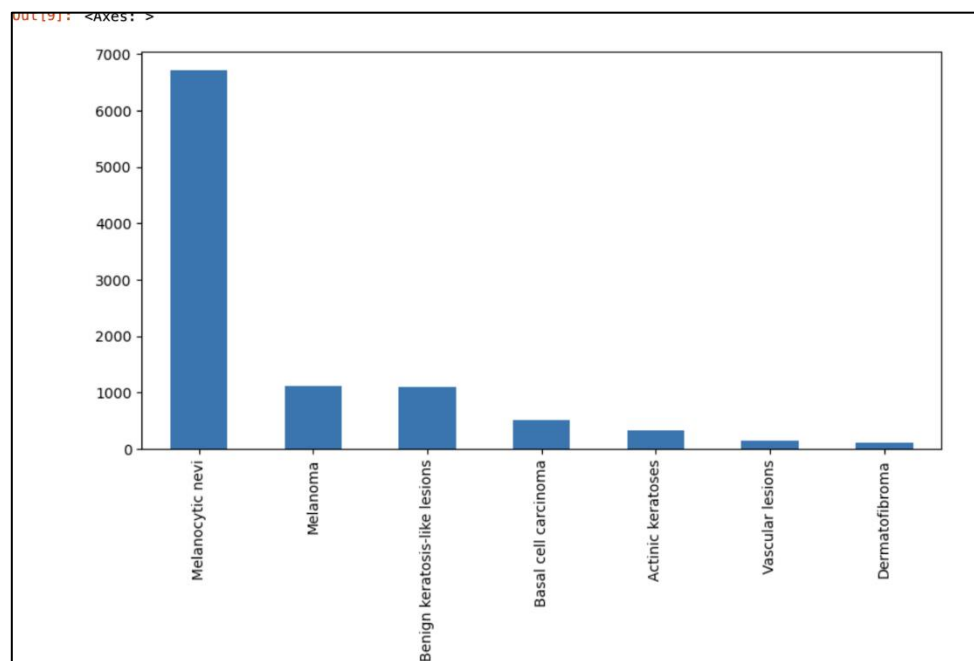


Figure. 19 : Bar graph for total number of images for each category

2) Total count of images for each diagnosis

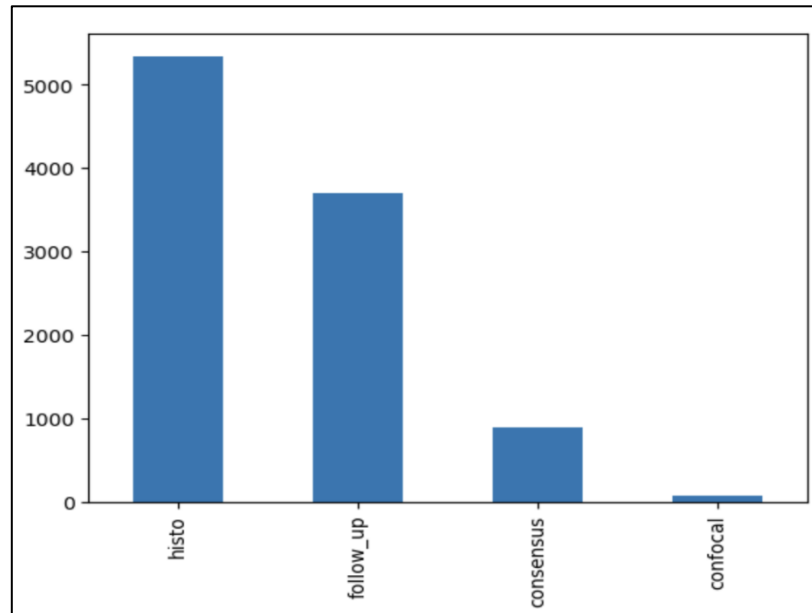


Figure. 20 : total count of images for each diagnosis

3) Total Count of images each location of the cancer

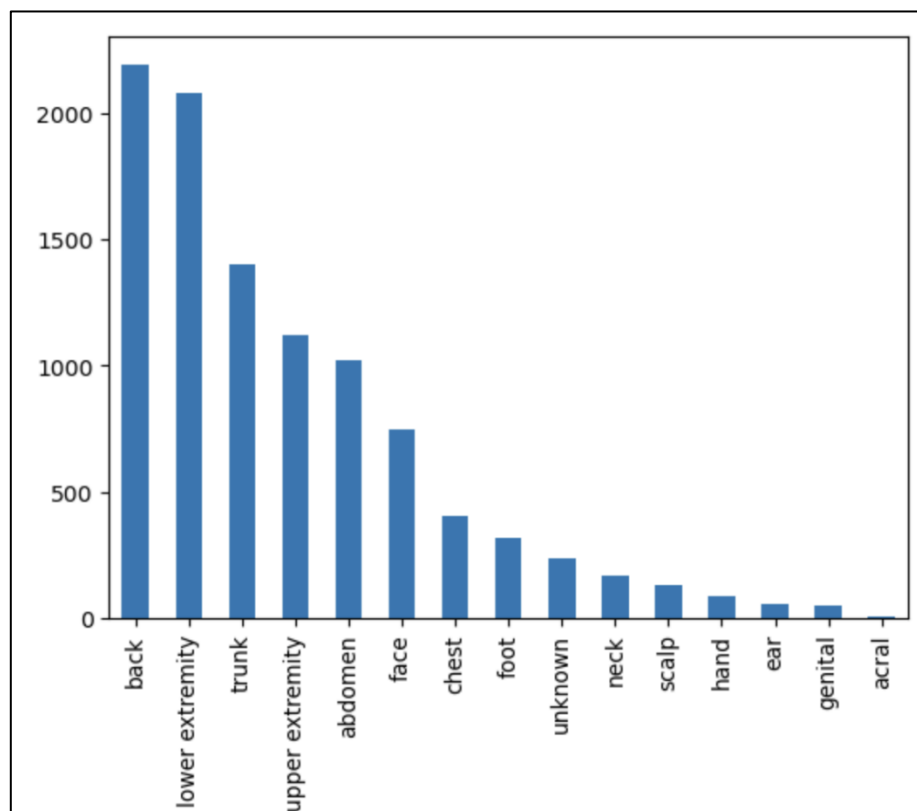


Figure. 21 : Bar graph for total images for each location of the cancer

4) Total count of images for distribution of age

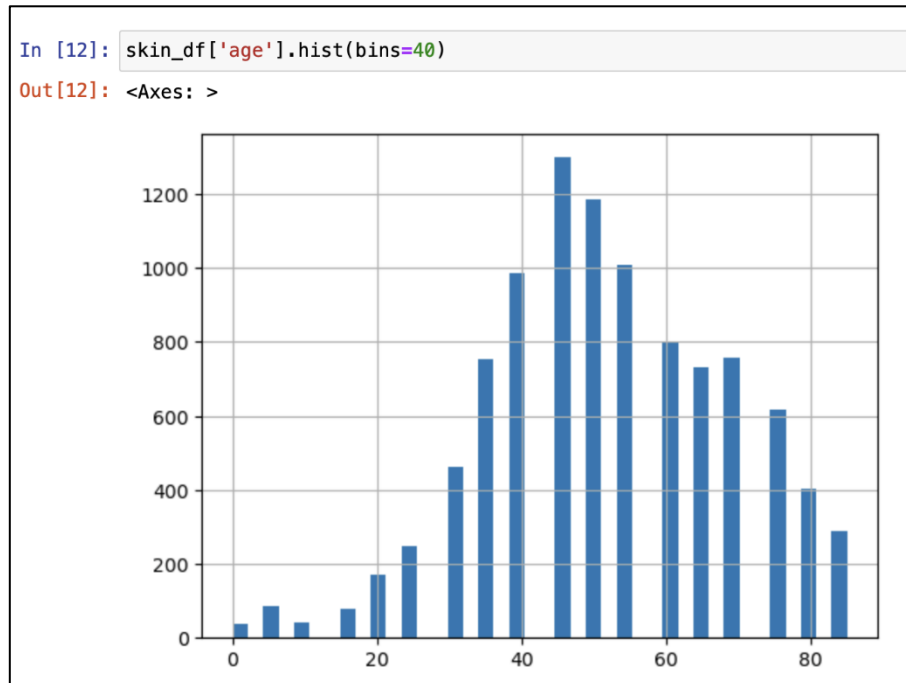


Figure 22 : Histogram for distribution of age

XgBoost classifier model results:

XGBoost (Extreme Gradient Boosting) is a powerful and widely used machine learning algorithm that belongs to the class of gradient boosting algorithms.

- Create a sagemaker session and S3 bucket for the session
- Create an IAM role to access the data
- Load the training and validation data in S3 bucket
- Training the model using container get_image_uri which will get the current region name using boto3.Session
- Deploy the model to perform inference(endpoint created)
- AWS Lambda- create a lambda function to invoke the endpoint
- Build a REST API to create a POST method to get the input in the form of pixels
- Test the Image to predict the class label

The components used for the model are:

Amazon S3 bucket for training and validation sets

- Amazon Sagemaker Notebook instance ml.t2.medium with 100 GB EBS and conda_pytorch_p39 kernel
- Amazon Sagemaker Pytorch
- HAM10000 dataset hosted by Harvard Dataverse

```

Xgboost_classifier1 = sagemaker.estimator.Estimator(container,
                                                    role,
                                                    instance_count = 1,
                                                    instance_type = 'ml.m5.xlarge',
                                                    output_path = output_location,
                                                    sagemaker_session = sagemaker_session, content_type='csv')

#We can tune the hyper-parameters to improve the performance of the model

Xgboost_classifier1.set_hyperparameters(max_depth=3,
                                       base_score=0.5, booster='gbtree',
                                       min_child_weight=1, num_round=100,
                                       n_estimators=100,
                                       n_jobs=-1, random_state=1855, reg_alpha=0,
                                       reg_lambda=1, scale_pos_weight=1,
                                       subsample=1,
                                       verbosity=0
                                       )

```

Figure 23: XGBoost Classifier Initialization and Hyperparameter Tuning in AWS SageMaker

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.39 | 0.24 | 0.30 | 29 |
| 1 | 0.58 | 0.33 | 0.42 | 45 |
| 2 | 0.51 | 0.43 | 0.47 | 86 |
| 3 | 0.00 | 0.00 | 0.00 | 5 |
| 4 | 0.79 | 0.96 | 0.87 | 504 |
| 5 | 1.00 | 0.10 | 0.18 | 10 |
| 6 | 0.52 | 0.15 | 0.24 | 72 |
| accuracy | | | 0.74 | 751 |
| macro avg | 0.54 | 0.32 | 0.35 | 751 |
| weighted avg | 0.70 | 0.74 | 0.70 | 751 |

Figure 24 : Classification Report for Skin Cancer Detection Model

MONAI using Pytorch:

The next architecture that we implemented is the MONAI framework with PyTorch. Medical Open Network for AI is a PyTorch-based open-source framework. We trained a DenseNet model with appropriate learning rate and epochs. (with limited computational power)

The components used in order to define the workflow of the classification were:

- Amazon S3 bucket for skin cancer images
- Amazon Sagemaker Notebook instance ml.t2.medium with 100 GB EBS and conda_pytorch_p39 kernel
- Amazon Sagemaker Pytorch managed container
- HAM10000 dataset hosted by Harvard Dataverse
- MONAI 0.3.0

The model performs decent in terms of accuracy, but much better in terms of f1 and recall. If we train it longer, it would surpass xgboost accuracy too.

```
In [ ]: estimator.fit({'train': inputs})

INFO: __main__: epoch 10 average loss: 1.0466
INFO: __main__: saved new best metric model
INFO: __main__: current epoch: 10 current AUC: 0.9284 current accuracy: 0.6887 best AUC: 0.9284 at epoch: 10
INFO: __main__: train completed, best_metric: 0.9284 at epoch: 10
INFO: __main__: Saving the model.
INFO: __main__:
      precision    recall  f1-score   support
akiec    0.5672    0.7454    0.6442     487
bcc      0.6495    0.5738    0.6093     549
bkl      0.5617    0.3982    0.4660     560
df       0.8182    0.9101    0.8617     534
mel      0.5387    0.6107    0.5725     524
nv       0.7657    0.6182    0.6841     571
vasc     0.9057    0.9963    0.9489     540
accuracy                0.6900    3765
macro avg    0.6867    0.6932    0.6838    3765
weighted avg    0.6887    0.6900    0.6832    3765

2023-12-04 16:41:07 Uploading - Uploading generated training model
2023-12-04 16:41:07 Completed - Training job completed
```

Figure 25 : Model Training Output with Performance Metrics

Before training the model, we process the data using MONAI pre-processing transforms. Integration with the MONAI framework by extending a SageMaker managed PyTorch container, The trained clusters are uploaded to S3. Once the model is trained, it is deployed to a SageMaker real-time endpoint for hosting on the instance. For inferences, we pass the validation dataset to the endpoint via the SageMaker API which returns the model predictions for class and probability. Below is the screenshot of, making predictions using the endpoint reference in AWS Sagemaker.

```
jupyter monai_skin_cancer Last Checkpoint: Last Monday at 11:52 (autosaved)
Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted conda_python3
+ - - - - - Run - - - - - Code - - - - - git nbdiff

# Define val_loader
val_loader = DataLoader(val_ds, batch_size=1, num_workers=1)

# Inference code
print('Sample Inference Results By Class:')
for i, val_data in enumerate(val_loader):
    response = predictor.predict(val_data[0])
    actual_label = val_data[1]
    pred = torch.nn.functional.softmax(torch.tensor(response), dim=1)
    top_p, top_class = torch.topk(pred, 1)
    print('actual class: ' + class_names[actual_label.numpy()[0]])
    print('predicted class: ' + class_names[top_class])
    print('predicted class probability: ' + str(round(top_p.item(), 2)))
    print()

Sample Inference Results By Class:
actual class: akiec
predicted class: vasc
predicted class probability: 0.8

actual class: bcc
predicted class: vasc
predicted class probability: 0.98

actual class: bkl
predicted class: nv
predicted class probability: 0.69
```

Figure 26 : Sample Model Inference Output on Validation Data

6. Future Work:

- **Increased Computational Power** : Invest in more powerful computing resources to handle larger datasets and more complex models. This could involve utilizing high-performance GPU instances on AWS for faster model training and inference.
- **Web Application Optimization** : Improve the user interface and experience of the web application. This could include developing a more intuitive design, smoother navigation, and quicker load times, particularly when processing images for classification.
- **Integration with Mobile Platforms** : Extend the project's reach by creating mobile applications compatible with iOS and Android. This would allow users to take photos of skin lesions and receive instant analysis through their smartphones.
- **Enhanced Data Processing Pipelines** : Optimize data preprocessing and feature extraction techniques to improve model performance. This could involve advanced image processing methods to better prepare images for classification.
- **Scalable Web Service Deployment**: Utilize AWS Lambda for deploying the web application to ensure scalability and cost-effectiveness. This serverless computing service can automatically manage the computing resources required by the application.
- **Continuous Model Improvement**: Implement continuous learning mechanisms where the model can learn from new data and user feedback. This could involve a system where dermatologists can contribute to the model's training by providing expert annotations.
- **Robust Security Measures**: Ensure that the web application and backend services are secure. This includes implementing best practices in web security, data encryption, and compliance with healthcare data regulations.
- **Automated Performance Monitoring**: Use AWS CloudWatch for automated monitoring of the application's performance. This can help in quickly identifying and addressing issues, ensuring high availability and reliability of the service.

These enhancements aim to leverage advanced computational resources and web technologies to improve the project's efficiency, accessibility, and overall impact in aiding early skin cancer detection.

REFERENCES:

1. <https://www.kaggle.com/datasets/andrewmvd/isic-2019>
2. <https://www.sciencedirect.com/science/article/pii/S2352914821001465>
3. <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9139937>
4. [AWS SageMaker Inbuilt Model Tuning](#)
5. [AWS API Gateway Integration with AWS Lambda and AWS SageMaker](#)
6. <https://github.com/aws/amazon-sagemaker-examples/tree/main>
7. <https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-algo-docker-registry-paths.html>

8. https://sagemaker.readthedocs.io/en/stable/frameworks/pytorch/using_pytorch.html
9. <https://github.com/Project-MONAI/MONAI>
10. <https://aws.amazon.com/blogs/industries/build-a-medical-image-analysis-pipeline-on-amazon-sagemaker-using-the-monai-framework/>
11. <https://github.com/zinebzannouti/AWS-Breast-Cancer-Prediction/tree/main>