CASE STUDY - IOT-BASED SKIN CANCER DETECTION APPLICATION

Submitted by	: Ramya Mercy Rajan
Roll Number	:AA.SC.P2MCA2107434
Subject	: IOT for AI
University	:Amrita Vishwa Vidyapeetham
Submitted to	: Dr. SS Chakravarthy

1. INTRODUCTION

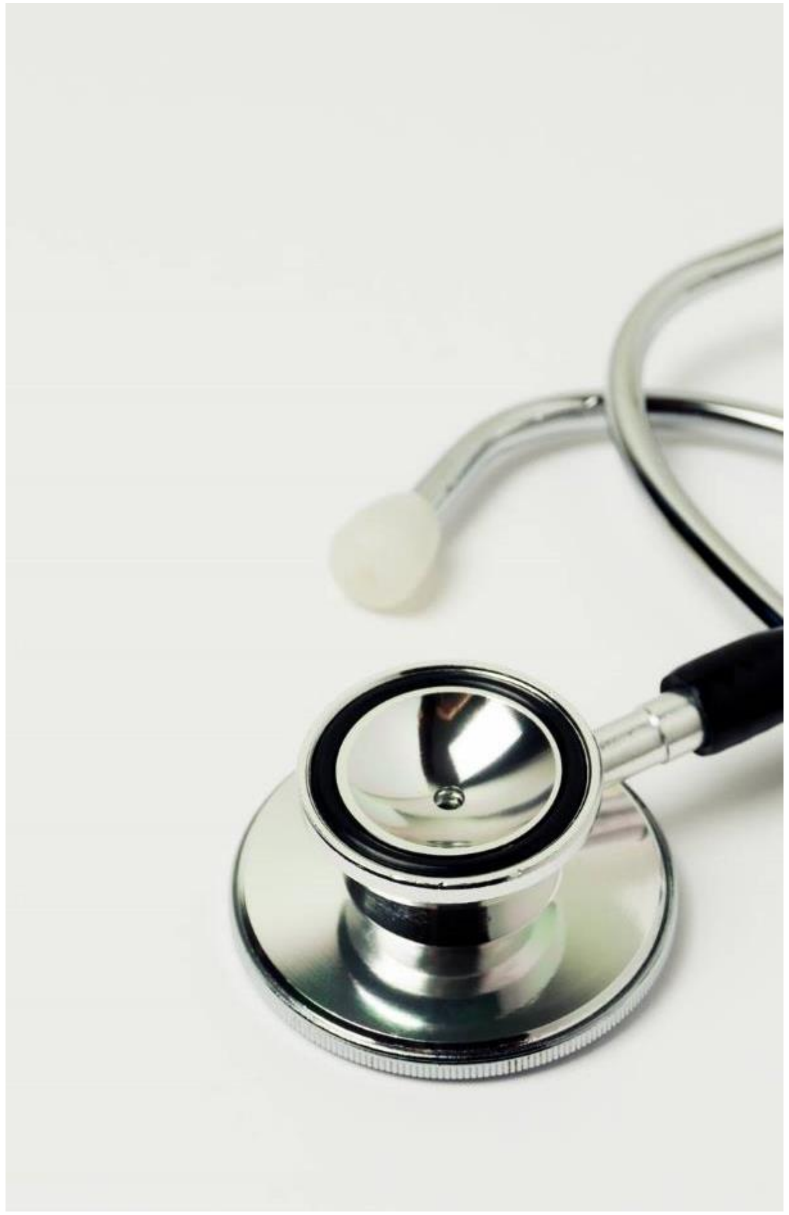
Background Information

Skin cancer is the most common cancer in the United States. It is estimated that approximately 9,500 people in the U.S. are diagnosed with skin cancer every day. More than three million cases of basal cell carcinoma are diagnosed in the United States every year, and nearly 20 Americans die from melanoma every day.

Melanoma and nonmelanoma are the two main types of skin cancer. Nonmelanoma is of lesser concern since it usually can be cured by surgery and is nonlethal. Melanoma, however, is the most dangerous skin cancer type.

Despite advances in therapeutics, the factors that most impact prognosis remain early recognition and removal of neoplasms before deep invasion or metastatic disease can occur.

With the increasing incidence of skin cancers, low awareness among a growing population, and a lack of adequate clinical expertise and services, there is an immediate need for AI systems to assist clinicians in this domain.



SKIN CANCER DETECTION

My aim is to predict the given skin lesion image taken by dermatoscopy and help medical professionals for early diagnosis and treatment of skin cancers. To train the deep learning model, I have used DermaMNIST by MedMNIST; this dataset contains multi-class images of skin lesions taken by dermatoscopy from real-life patients. It has 10000 images with seven classes that could be divided into malignant (Benign keratosis-like lesions, Dermatofibroma, Melanocytic nevi, Vascular lesions) and benign groups(Actinic keratoses and intraepithelial carcinoma, Basal cell carcinoma, Melanoma). Skin cancer detection using artificial intelligence (AI) is an innovative IoT application aimed at providing an efficient and accessible way to screen for skin abnormalities that could be potential signs of skin cancer. In this case study, I'll discuss the development of an IoT-based skin cancer detection system. The project involves a hardware circuit, IoT setup, machine learning techniques, its practical applications, and the results achieved.

2. PROJECT EXECUTION

2.1 Data Acquisition

I will be utilizing the DermaMNST by MedMNIST. This is a dataset containing multiclass images as (28 x 28) (2D) numpy arrays in the form of an npz file. The images in the dataset have been collected from real life patients via dermascope. The dataset was downloaded from the official website: Available at <https://medmnist.com/>

2.2 Training Methodology

It is a vast dataset of 10,015 pictures of skin lesions which can be used to identify early stages of skin cancer by classifying them as malignant or benign. This data set is divided into training, validation and testing dataset; 7007, 1003 and 2005 pictures respectively. I will be building a deep learning neural network which will process an image and give a prediction output.

2. PROJECT EXECUTION

- Model Architecture

The model will be built using keras and tensorflow version 2.7.0. The model is the Convolutional Neural Network which is often used to analyze visual input. The model is made up of several types of layers like Flatten, Dense, Dropout, Batch Normalization, Pooling and 2D convolutional layers using activation functions 'relu' and 'softmax'.

I have created a sequential model which is a deep learning model where model layers are created and added to sequentially.

The 2D Convolutional layer is the main layer in the CNN which contains a set of kernels which help the model learn parameters through training. The filters convolve with the input images to create a feature map that is used to see which regions in the input image are relevant to the class.

The Batch Normalization layer is used to standardize the inputs to a layer for each mini-batch. This will help stabilize the learning process, and reduce the number of training epochs required to train deep networks.

The Pooling Layer is added to reduce the dimensions of the feature maps. It reduces the number of parameters to learn and the amount of computation performed in the network. The pooling layer summarizes the features present in a region of the feature map generated by a convolution layer.

2. PROJECT EXECUTION

- The Flatten Layer will be converting the data into a 1D array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector.
- The Dropout Layer will be used to prevent the possibility of overfitting of the model by randomly setting input units to 0 with a frequency of rate at each step during training time.
- The Dense Layer will ensure that all outputs from the previous layer are input to all its neurons, each neuron providing one output to the next layer. It is the fully connected layer in Keras and will have the neurons and activation function specified in this layer.
- The activation function is used to apply a certain degree of nonlinearity into the output of a neuron. The softmax function is used as the activation function in the output layer of neural network models that predict a multiclass probability.
- The ReLU is a function that will output the input directly if it is positive, otherwise, it will output zero. It will prevent the vanishing gradient problem, allowing models to learn faster and perform better

PROJECT EXECUTION

Model Training

- The model will be trained with an optimum number of epochs to achieve a high accuracy. The epochs determine the number of times the dataset will run through the CNN to train. If there are way too many epochs it could result in overfitting hence it is best to use an optimal number of epochs with early stopping call back function which will stop the training once the model performance stops improving on a hold out validation dataset.
- The model will be trained with a low learning rate which will ensure that the model is able to learn optimally by globally setting suitable weights. This will take longer to train but it will produce better results than if the learning rate is high.
- The model performed better with focal loss function which could assist in tackling the issue faced by the imbalanced dataset. Optimizers are functions that are used to modify the attributes such as weights and learning rate of a neural network. This assists in reducing the overall loss and improves the accuracy.
- The model will utilize Adam Optimizer by Keras. This algorithm will take into consideration the 'exponentially weighted average' of the gradients and use this concept to get an average to accelerate the gradient descent.

PROJECT EXECUTION

- Focal Loss

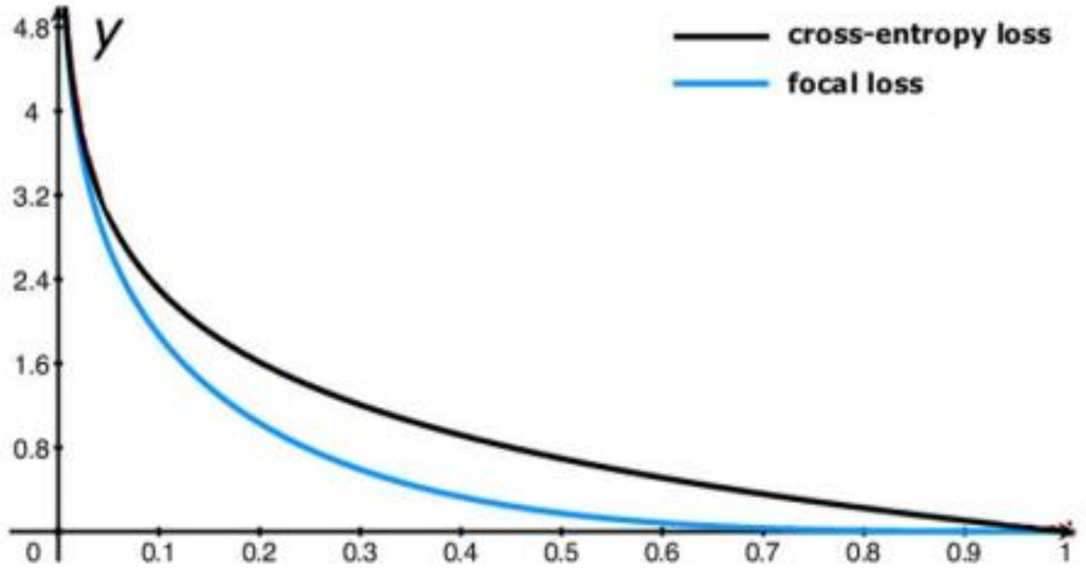
Unbalanced data like what I have here can cause some problems to the model learning procedure. To address this problem, I have used a new loss function named focal loss. With the focal loss, we add a modulating factor multiplied by the Cross-Entropy Loss. By this modulating factor, when a sample is misclassified, p is low and modulating factor is near one, so the loss is unaffected. But as p increases, the modulating factor approaches 0, and the loss for well-classified examples is down-weighted.

$$\mathrm{FL}(p_t) = -\alpha_t (1 - p_t)^\gamma \log(p_t).$$

PROJECT EXECUTION

$$\mathrm{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t).$$

This modulating factor reduces the loss function contribution from easy examples and extends the range in which an example receives low loss. So the model will correctly learn the data and save from overfitting.



2.3 SERVICE IMPLEMENTATION

- A “Benign” tumor refers to a condition, tumor, or growth that is not cancerous and it means that it does not spread to other parts of the body. It does not invade nearby tissue. A “Malignant” tumor means it's cancerous. Malignant tumors contain cells that are cancerous, growing out of control and capable of metastasizing. Metastasize denotes that the cells of the tumor are able to leave the original tumor and travel to other parts of the body.
- Intervention and stopping the cancer growth is now required. Research indicates that most experienced physicians can diagnose cancer with 76% accuracy. I have executed all these images in order to specify if that cancer out of these images is malignant or benign. I add up all these features into our machine learning model and the model teaches the machine how to basically classify images or classify data and report us if it is malignant or benign. This data set is divided into training, validation, and testing dataset. I have built a deep learning neural network which processed an image and gave a prediction output.
- The model is the Convolutional Neural Network which is often used to analyze visual input. The model is trained with an optimum number of epochs to achieve a high accuracy.

OUTCOMES

- 3.1. Data Preprocessing

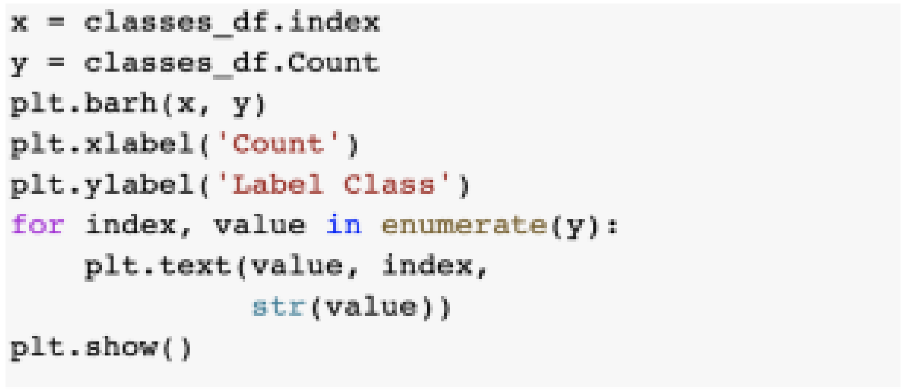
All images in our dataset were pre-processed into (28 x 28 x 28) (3D) NumPy arrays in the form of an npz file. However, given that the original images consist of RGB coefficients in the range of [0, 255], more importantly, the sample size and the imbalanced nature of outcomes, i.e., the distribution of classes in our dataset. I have implemented data pre-processing and augmentation methods concerning the data role. These methods were applied for all training, validation, and testing datasets.

Govern by the domain knowledge, model performance and accuracy all of the following were applied for the training images using ImageDataGenerator API from tensorflow :

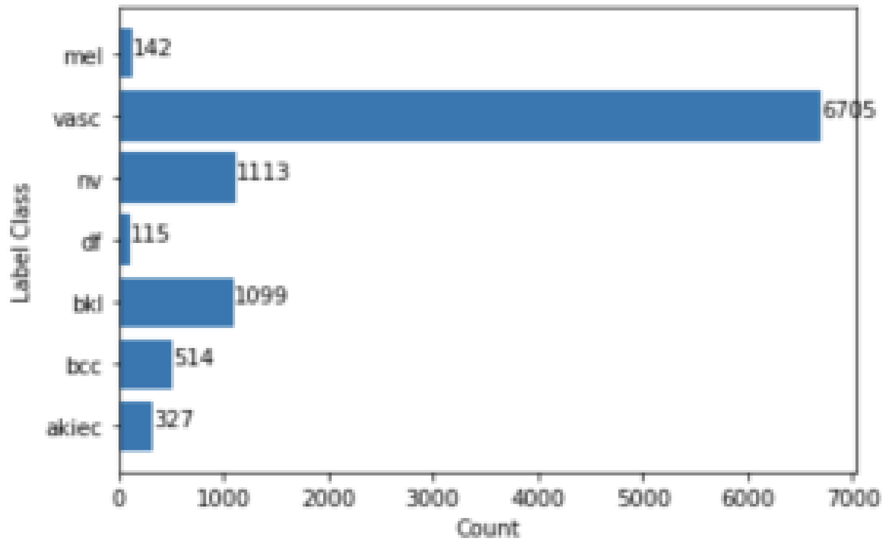
1. Rescaling: Rescaled all images from [0, 255] to [0, 1].
2. 90 degree range for random rotations.
3. Random horizontal and vertical flip.
4. Random zoom range of 10%.
5. Width and height shift of 10% of the total width/height

Only rescaling was implemented to the validation and testing dataset. Data augmentation is also a very important aspect that will assist in preventing the overfitting of the model by increasing the number of data present in the training dataset.

OUTCOME



```
x = classes_df.index
y = classes_df.Count
plt.barh(x, y)
plt.xlabel('Count')
plt.ylabel('Label Class')
for index, value in enumerate(y):
    plt.text(value, index,
             str(value))
plt.show()
```

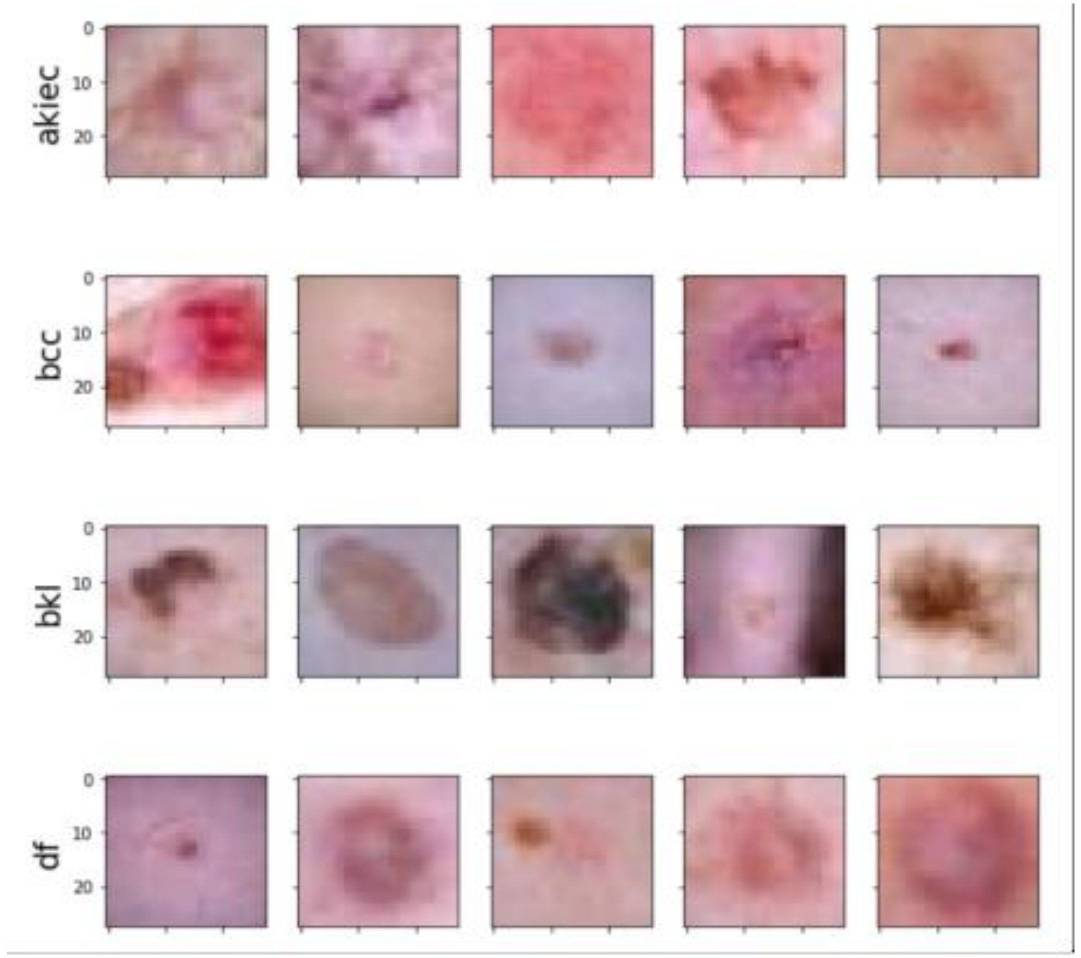


- 3.2 Exploratory Data Analysis (EDA)

The data has been analyzed using EDA to understand how to use the details to our advantage and summarize the vast data provided to us. I first checked the files and their shape and size. Using numpy unique modules we were able to check how many classes we would be dealing with in the dataset.

I have concatenated the X and Y datasets to check the frequency of the data. I tried plotting the data frame of frequency of each class with a matplotlib graph. This graph showed the imbalance in the dataset with one class having over 6000 images while the others had fewer images.

OUTCOME



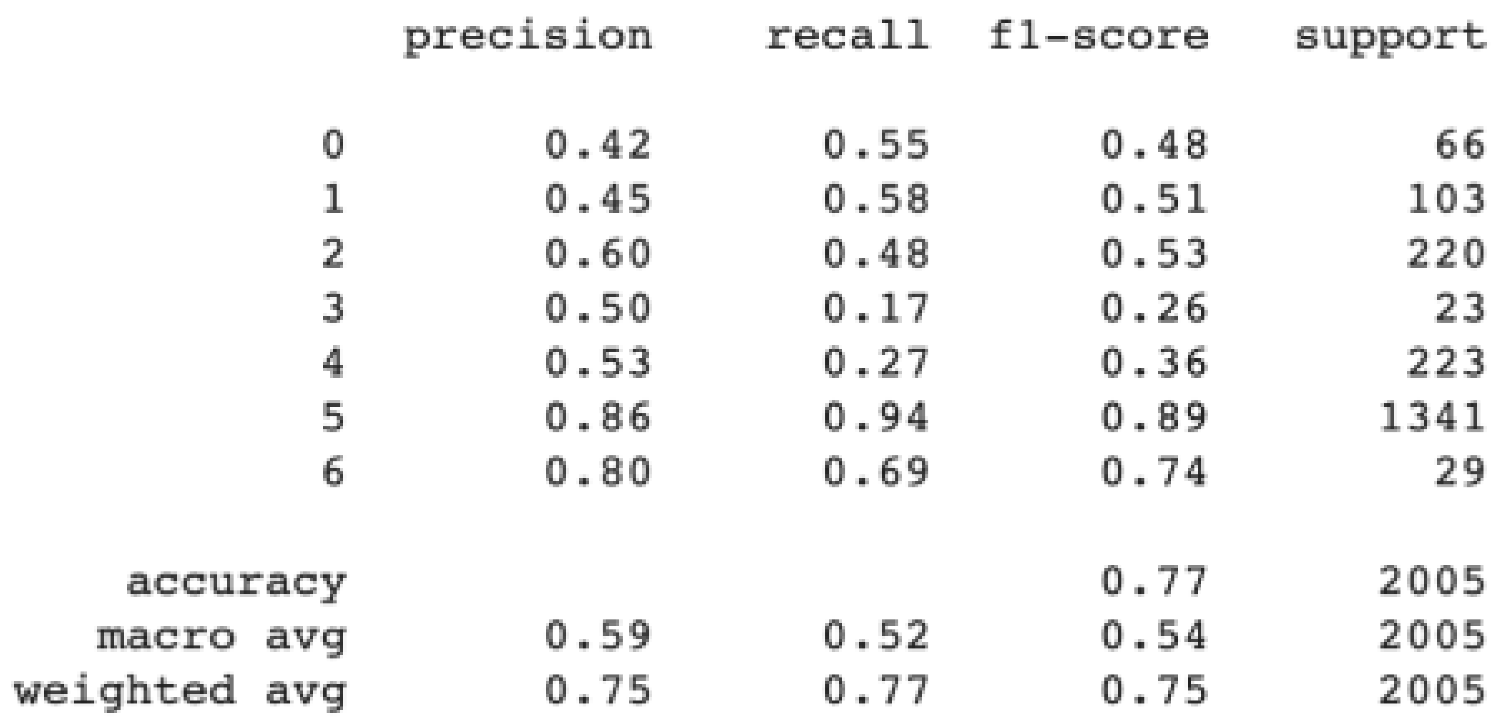
- I also plot a grid to visualize the images in the dataset according to the multiclass labels to get an overview of what kind of images I will be dealing with.

3.3 MODELING

- The model was trained rigorously to ensure that we achieved the highest possible accuracy. I've used an early stopping function with a callback to stop model training when the highest possible accuracy had been attained. In a medical problem, accuracy is not the only important metric that needs to be evaluated.
- The precision and recall of the model is of equal importance, hence I had to ensure that the precision, recall and F1 score of all classes were not zero. After multiple attempts I achieved an accuracy of 77% with a decent precision and recall for all classes.
- Previously, I was not able to get precision and recall more than zero for the classes with less images. After attempting with focal loss we achieved a better figure.

OUTCOME

- Below is the classification report of the model which was achieved after multiple attempts:



```
              precision    recall  f1-score   support

           0       0.42      0.55      0.48        66
           1       0.45      0.58      0.51       103
           2       0.60      0.48      0.53       220
           3       0.50      0.17      0.26        23
           4       0.53      0.27      0.36       223
           5       0.86      0.94      0.89      1341
           6       0.80      0.69      0.74        29

    accuracy                           0.77      2005
   macro avg       0.59      0.52      0.54      2005
weighted avg       0.75      0.77      0.75      2005
```

OUTCOME

- 3.5. Testing and Improvements.

The dataset was split into 70%, 10%, and 20% for training, validation and testing; respectively, before several models were trained with different combinations of hyperparameters. The final model was tested using the 20% of the dataset, which is unseen data that the algorithm never previously encountered.

In Google Colab I printed the classification report and the summary of the training and validation to check the results of the model. In all experiments and due to the unbalanced nature of the dataset, the accuracy (ACC) score and F1-score were used as performance metrics.

In future studies, this model accuracy can be improved by trying one or all of the following; first, include more images for the under sampled classes. This will help tackle the imbalance dataset issue that could be a bottleneck while classifying images that have very few images in the training phase of the model training.

In this study other pretrained models can be tried e.g., U net, ResNet and more using supportive hardware such as faster processor and GPU.

The project involves a hardware circuit, IoT setup, machine learning techniques, its practical applications, and the results achieved.

1. Hardware Circuit:
The hardware component of this project consists of a Raspberry Pi with a camera module. The Raspberry Pi serves as the central processing unit for capturing high-resolution images of the skin. It is equipped with a camera module that can capture images with sufficient detail for analysis. The camera module connects to the Raspberry Pi's GPIO pins.

2. IoT Setup: The IoT setup involves connecting the Raspberry Pi to the internet and setting up a secure connection to transmit captured images to a remote server for analysis. This includes the following steps:
•**Raspberry Pi Configuration:** Ensure the Raspberry Pi is set up with the necessary operating system, libraries, and drivers to operate the camera module.
•**Internet Connectivity:** Connect the Raspberry Pi to the internet, either via Wi-Fi or Ethernet.
•**Image Transmission:** Develop a Python script that captures images at regular intervals and sends them to a remote server for analysis.
•**Server Setup:** Configure a remote server that receives incoming skin images from the Raspberry Pi, runs the image analysis, and returns the results.

3. Learning Techniques: The heart of this application is the AI-based skin cancer detection model. Here's how it works:
•**Image Classification Model:** A deep learning model, often based on convolutional neural networks (CNNs), is trained to classify skin images into different categories, including normal, benign, and malignant. The model is trained on a large dataset of skin images with known diagnoses.
•**Transfer Learning:** Transfer learning is often used to leverage pre-trained models, such as those trained on ImageNet, and fine-tune them for skin cancer detection.
•**Model Training:** The AI model is trained to identify various visual cues associated with skin cancer, including irregular shapes, uneven color distribution, and the presence of specific lesions.
•**Real-time Analysis:** As images are captured by the Raspberry Pi and transmitted to the server, the AI model runs real-time analysis to classify them.

In this project it utilizes IoT and hardware circuits, the focus is on acquiring skin images or relevant data, processing that data using IoT hardware, and applying machine learning techniques to detect potential skin cancer.

a) Data Acquisition:

•**Imaging Sensors:** Utilize cameras or specialized dermatoscopic imaging devices to capture high-resolution images of the skin lesions.

•**Spectrophotometers:** In some cases, spectrophotometers may be used to analyze the spectral characteristics of skin lesions.

b) Microcontrollers or Single-Board Computers:

•Use of microcontroller Raspberry Pi to interface with imaging sensors. These devices have GPIO pins that can be used to connect camera.

c) Data Preprocessing:

•Prepare the acquired images by performing preprocessing tasks, such as resizing, color correction, and noise reduction. This ensures the input data is suitable for machine learning.

d) Machine Learning Algorithms:

•Train machine learning models, often based on deep learning techniques, to analyze skin images. Convolutional Neural Networks (CNNs) are commonly used for image analysis tasks.

•Supervised Learning: Train models to classify skin lesions as benign or malignant based on labeled image data.

•Transfer Learning: Leverage pre-trained models (e.g., ImageNet) and fine-tune them for skin cancer detection.

•Data Augmentation: Increase the size of the training dataset by applying transformations like rotation, flipping, and scaling to images.

•Ensembles: Create ensemble models to improve accuracy and reliability.

e) Edge Computing:

1.**Data Capture:** IoT devices equipped with cameras capture skin images or relevant data from patients.
2.**Data Preprocessing:** Perform initial data preprocessing on the edge devices, such as resizing, filtering, and removing noise from the images.
3.**Inference:** Run machine learning models for initial image analysis directly on the edge devices. Inference at the edge allows for real-time decision-making and minimizes the need to send large volumes of data to the cloud.

Cloud Computing (AWS):

f) Data Transfer: Send processed data, including analyzed results and images, to the AWS cloud for storage and further analysis.

- **Data Storage:** Store all data securely in AWS S3 buckets, a scalable and reliable cloud storage service. Set up a specific S3 bucket for storing skin images and another for diagnostic results.
- **Data Analysis:** Use AWS Lambda to automatically process incoming data and trigger additional actions. We can even automate notifications to healthcare professionals based on the diagnostic results.
- **Machine Learning in the Cloud:** Train and fine-tune machine learning models in the cloud using services like Amazon SageMaker. These models can be periodically updated based on new data.

- **Continuous Learning:** Implement reinforcement learning techniques in the cloud to enable the system to improve its diagnostic accuracy over time.
- **Security and Compliance:** Leverage AWS Identity and Access Management (IAM) and other security services to protect data privacy and comply with healthcare regulations.
- **Scalability:** AWS services, like AWS EC2 for compute and AWS RDS for databases, ensure the system can scale to handle a growing number of patients and devices.
- **Monitoring and Alerts:** Implement AWS CloudWatch for monitoring the health and performance of AWS resources. Set up alarms to receive notifications about any issues.
- **Data Backup:** Configure automatic data backups and redundancy in AWS to ensure data integrity and availability.
- By combining edge computing for real-time processing on IoT devices with cloud computing through AWS, we can achieve a system that provides quick, real-time diagnostic results while taking advantage of the scalability, reliability, and advanced machine learning capabilities of AWS cloud services. This approach is well-suited for healthcare applications, where real-time decision-making is critical, and centralized data storage and analysis are also required for long-term trends and continuous learning.

g) Connectivity:

•Implement connectivity options to facilitate model updates, maintenance, and remote monitoring. Wi-Fi or cellular connections are common choices.

h) Power Management:

•Optimize power management for battery-operated devices to extend device lifetimes.

i) Security:

•Implement data encryption and device security to protect sensitive health data.

j) Continuous Learning:

•Consider using reinforcement learning techniques to allow the system to learn from new data continuously and improve its diagnostic accuracy.

k) Data Storage:

•Store historical data and diagnosis results. Optionally, use cloud storage for backup and analysis.

l) Cloud Integration:

•Integrate with cloud platforms to centralize and aggregate data from multiple IoT devices for population-level analysis and continuous model updates.

m) Scalability:

•Ensure the system can scale to handle a growing number of IoT devices as the project expands.

In a skin cancer detection project, these learning techniques help in automating the process of diagnosing skin lesions, allowing for early detection of potential issues. It's crucial to continually validate and update the models with new data to enhance their accuracy and ensure patient safety. Additionally, privacy and security should be top priorities, given the sensitive nature of health-related data.

4. End Usage: The end usage of this skin cancer detection IoT application is multi-faceted:

•**Early Skin Cancer Detection:** The primary goal is to detect skin cancer at an early stage when it is most treatable. Users can regularly capture images of their skin using the Raspberry Pi.

•**Remote Consultation:** Users can share their skin images with dermatologists or healthcare providers for remote consultation, especially useful in areas with limited access to healthcare.

•**Data Collection:** The system allows for large-scale data collection and analysis, contributing to research on skin cancer diagnosis and treatment.

•**Education and Awareness:** The application can serve as an educational tool, helping individuals learn about skin cancer and its early signs.

5. Results: The success of this IoT-based skin cancer detection application is measured by its accuracy in identifying potential skin abnormalities. The results are evaluated based on sensitivity, specificity, and overall accuracy. Additionally, the application's usability, accessibility, and impact on early skin cancer detection in real-world scenarios are essential factors to consider.

Skin Cancer Detection Project - Deployment Report

Comprehensive guide for deploying a skin cancer detection system, combining local processing on a Raspberry Pi with the scalability and reliability of AWS cloud services.

1. Raspberry Pi Setup:

Ensure that our Raspberry Pi is set up and running. Make sure it's connected to the internet and can communicate with external services.

2. Python Flask Web Application:

Create a Python Flask web application for skin cancer prediction. This application will handle incoming image uploads and return predictions.

```
from flask import Flask, request, jsonify

import numpy as np

from tensorflow.keras.models import load_model

from PIL import Image

import io
```

```
app = Flask(__name)
model = load_model('SkinSavior_model.hdf5')

@app.route('/predict', methods=['POST'])
def predict_skin_cancer():
    try:
        image = request.files['image'].read()
        image = Image.open(io.BytesIO(image))
        image = image.resize((28, 28))  # Resize the image to match your model's input size
        image = np.array(image) / 255.0
        prediction = model.predict(np.expand_dims(image, axis=0))
        predicted_class = np.argmax(prediction)
        class_names = ["Class 0", "Class 1", "Class 2", "Class 3", "Class 4", "Class 5", "Class 6"]
```

```
result = {"class_name": class_names[predicted_class], "prediction": float(prediction[0][predicted_class])}
        return jsonify(result)
    except Exception as e:
        return jsonify({"error": str(e)})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)  # runs the Flask app on port 80 for external access
```

3. Web Server Setup:

In the skin cancer detection project, setting up a web server is crucial for deploying our Flask-based skin cancer prediction application. This report provides a detailed guide on configuring Nginx as a reverse proxy to route incoming HTTP requests to our Flask app running on a specific port, such as port 80.

Requirements:

Before proceeding with the web server setup, ensure that the following requirements are in place:

Raspberry Pi: Raspberry Pi should be correctly configured and connected to the internet.

Python Flask Application: Python Flask application should be developed and tested for skin cancer prediction as per the code provided above.

Web Server Setup:

Step 1: Nginx Installation

If Nginx is not already installed on Raspberry Pi, install it using the package manager:

```
sudo apt update
sudo apt install nginx
```

Step 2 : Create Nginx Configuration

Create a configuration file for our Flask application in the Nginx sites-available directory:

```
sudo nano /etc/nginx/sites-available/skin-cancer-app
```

Next stoep in the configuration file, define our server block to route incoming requests to our Flask app. Here is a sample configuration:

```
server {
    listen 80;
    server_name your_domain.com;  # Replace with your domain or IP
    location / {
        include uwsgi_params;
        uwsgi_pass unix:/path/to/your/app.sock;
    }
}
```

Step 3: Enable the Site Configuration

Create a symbolic link to enable our site configuration:

```
sudo ln -s /etc/nginx/sites-available/skin-cancer-app /etc/nginx/sites-enabled/
```

Step 4: Test Nginx Configuration

Checking the configuration for syntax errors:

```
sudo nginx -t
```

If there are no errors, we will be able to see an "OK" message.

Step 5: Restart Nginx

Now, restart Nginx to apply the new configuration:

```
sudo systemctl restart nginx
```

Step 6: Adjust Firewall Settings (if necessary)

If we have a firewall enabled, such as UFW, we may need to allow traffic on the port we're using (e.g., port 80):

```
sudo ufw allow 80/tcp
```

Ensure that our Raspberry Pi's firewall settings permit external access.

With these steps completed, Nginx is now configured as a reverse proxy to route incoming HTTP requests to our Flask app. Users can access our skin cancer detection application through our Raspberry Pi's domain or IP address. This setup is essential for enabling external access to our model and providing skin cancer prediction services.

4. Raspberry Pi and Camera Integration:

Ensure our Raspberry Pi is equipped with a camera module, and we have the necessary drivers and software to capture images.

Python script to capture images from the camera and send them to Flask app for prediction.

```
import requests
import picamera
camera = picamera.PiCamera()
def capture_and_predict_image():
    camera.capture('image.jpg')  # Capture an image
    files = {'image': open('image.jpg', 'rb')}
    response = requests.post("http://our-raspberry-pi-ip/predict", files=files)
    print(response.json())

if __name__ == '__main__':
    capture_and_predict_image()
```

Ensure that the model (SkinSavior_model.hdf5) is present on your Raspberry Pi.

5. Deploy Flask Application on AWS:

a. Set Up an AWS Account

b. Choose an AWS Service: There are several ways to deploy a Flask application on AWS. Some common options include using AWS Elastic Beanstalk, AWS Lambda, or AWS EC2.

c. Create a Virtual Environment: Within our chosen AWS service, we have to create a virtual environment to run our Flask application. Ensure that we install the necessary dependencies, including Flask and any required packages.

d. Upload Flask Application: Deploy our Flask application code to AWS. This may involve using the AWS Console, AWS CLI.

e. Configure Security: Set up security groups and configure firewall rules (e.g., using AWS Security Groups) to allow external access to our Flask application.

f. Domain Name: We can configure a domain name for our Flask application, either using AWS Route 53 or by linking our existing domain to AWS.

g. SSL Certificate: For secure communication, consider obtaining an SSL certificate for our domain name. AWS provides the AWS Certificate Manager service for this purpose.

6. Host Model on AWS Cloud Storage:

a. Choose AWS S3: Amazon Simple Storage Service (Amazon S3) is a reliable and scalable object storage service that we can use to host our model.

b. Upload Model to S3: Use the AWS CLI or AWS SDKs to upload our skin cancer detection model (ie, "SkinSavior_model.hdf5") to an S3 bucket. Ensure that the bucket's permissions are set to allow access to the model file.

c. Secure the Model: We can configure S3 bucket policies and permissions to control who can access the model. Make sure the model is not publicly accessible unless that is our intention.

7. Integrate Flask Application and Model:

a. In our Flask application, use the AWS SDK (Boto3 for Python) to interact with AWS services. We can use Boto3 to access the S3 bucket where our model is stored.

b. Load the model in our Flask application from the S3 bucket for skin cancer predictions.

With this setup, our Flask application on AWS will be able to access the skin cancer detection model hosted in an S3 bucket.

REFERENCES

- https://www.aad.org/media/stats-skin-cancer
- Ray A, Gupta A, Al A. Skin lesion classification with deep convolutional neural network: process development and validation. JMIR Dermatol. 2020 May 7;3(1):e18438. doi: 10.2196/18438
- Glazer AM, Rigel DS, Winkelmann RR, Farberg AS. Clinical Diagnosis of Skin Cancer: Enhancing Inspection and Early Recognition. Dermatol Clin. 2017 Oct;35(4):409-416. doi: 10.1016/j.det.2017.06.001. Epub 2017 Aug 7. PMID: 28886797
- Goyal M, Knackstedt T, Yan S, Hassanpour S. Artificial intelligence-based image classification methods for diagnosis of skin cancer:

REFERENCES

- Omania Health. 2019. Skin Cancer Growing Public Concern UAE. Omania Health. [online]. Available at”
- https://insights.omnia-health.com/medical-specialities/skin-cancer-growing-public-co ncern-uae
- Brownlee, J. 2019. “A Gentle Introduction to the Rectified Linear Unit (ReLU)”. Machine Learning Mastery. [online]. Available at:
- https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-lear ning-neural-networks/
- https://towardsdatascience.com/a-loss-function-suitable-for-class-imbalanced-data-fo cal-loss-af1702d75d75
- Deploying a Flask application to Elastic Beanstalk - AWS Elastic Beanstalk (amazon.com)
- Hands-On Tutorials for Amazon Web Services (AWS)
- Python, Boto3, and AWS S3: Demystified – Real Python
- nginx — Flask Documentation (2.1.x) (palletsprojects.com)
- nginx — Flask Documentation (2.1.x) (palletsprojects.com)

Conclusion:

The development of an IoT-based skin cancer detection system involves the integration of hardware, IoT, AI, and healthcare. It provides a convenient and accessible way for individuals to monitor their skin health and receive early warnings about potential skin cancer. Moreover, the project contributes to the field of AI in healthcare and telemedicine by enabling remote consultations and large-scale data collection for skin cancer research.

THANKYOU

Submitted by : Ramya Mercy Rajan

Roll Number : AA.SC.P2MCA2107434

Subject :IOT for AI

University :Amrita Vishwa Vidyapeetham

Submitted to : Dr. SS Chakravarthi