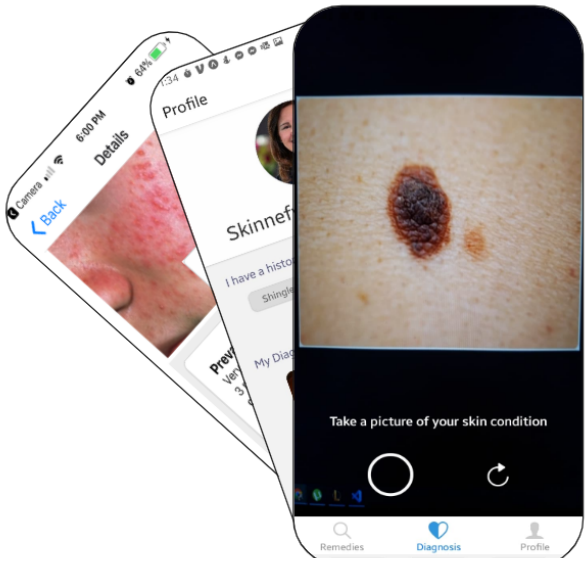
N



Skinnefy

A Spring 2020 Senior Design Project

Naman Pujari **Siddharth Rajan** **Harsh Patel** **Faheem Kamal**

H

Project Overview

Over the past couple semesters we…

- Created a skin condition detection application
- Developed and Deployed our very own Image Classification model

Along the way we…

- Considered a wide variety of technologies
- Learned a **lot** with regards to ML and App Development
- Discovered some of our strengths and weaknesses
- Got accustomed to today's top industry technologies

But before we go into the details

Here’s our Final Product

Diagnosis Tab

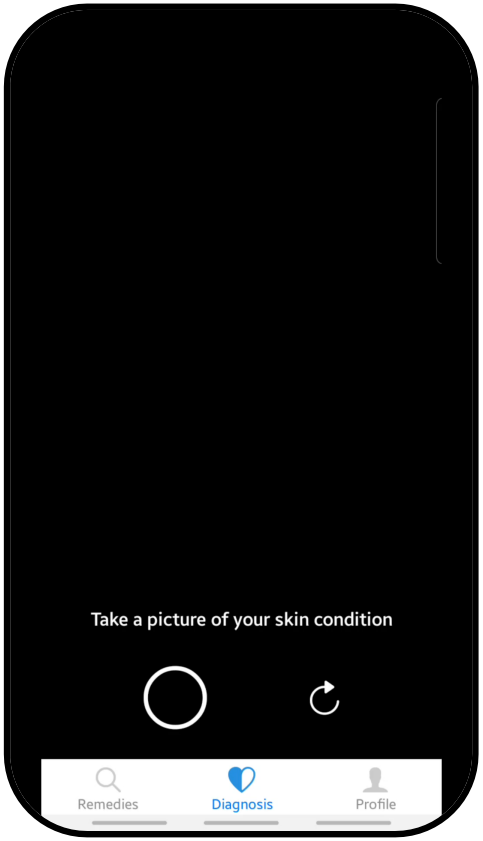
Allows users to take a picture and receive a diagnosis

Profile Tab

Displays all relevant user information, including previous diagnoses

Remedies Tab

An informative knowledge base pertaining to skin conditions that *Skinnefy* may diagnose users with



F

Motivation

- We wanted to provide a proper and convenient solution that is available to consumers worldwide on any iOS and Android device.

F

Background

- Skin conditions affect an estimate of around 1.9 billion people.
- Access of Dermatology care is limited.
- Store-and-forward tele-dermatology has become popular.

F

Related Works

- Aysa
- FirstDerm AutoDerm
- DLS for Differential Diagnosis of Skin Disease

F

Theory

F

Image Classification

A complex process in Computer Vision that can classify an image according to its visual content.

Two main classification methods are Supervised classification and Unsupervised classification.

Convolutional Neural Networks

- CNN is a class of deep neural networks.
- It is a feedforward neural network where units are organized into layers.
- Very effective in Image recognition and classification.

H

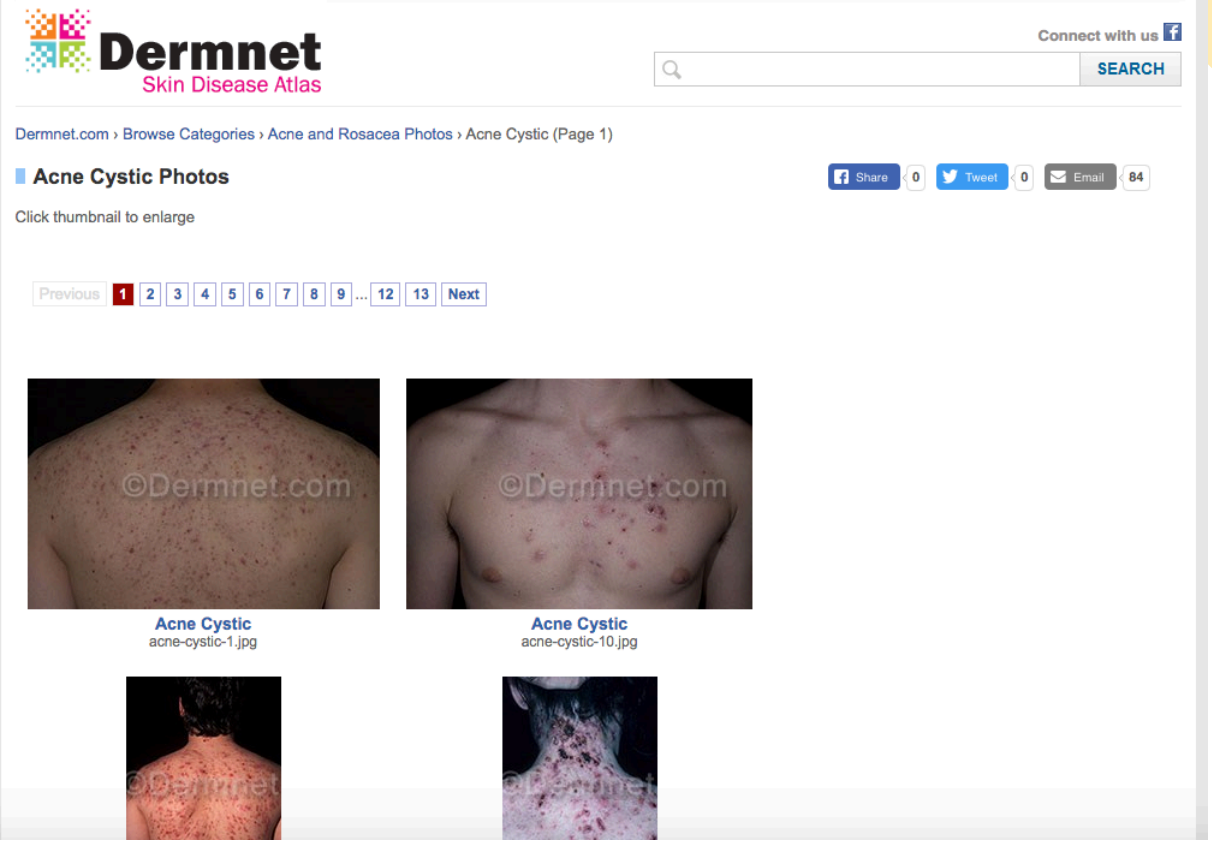
Datasets

H

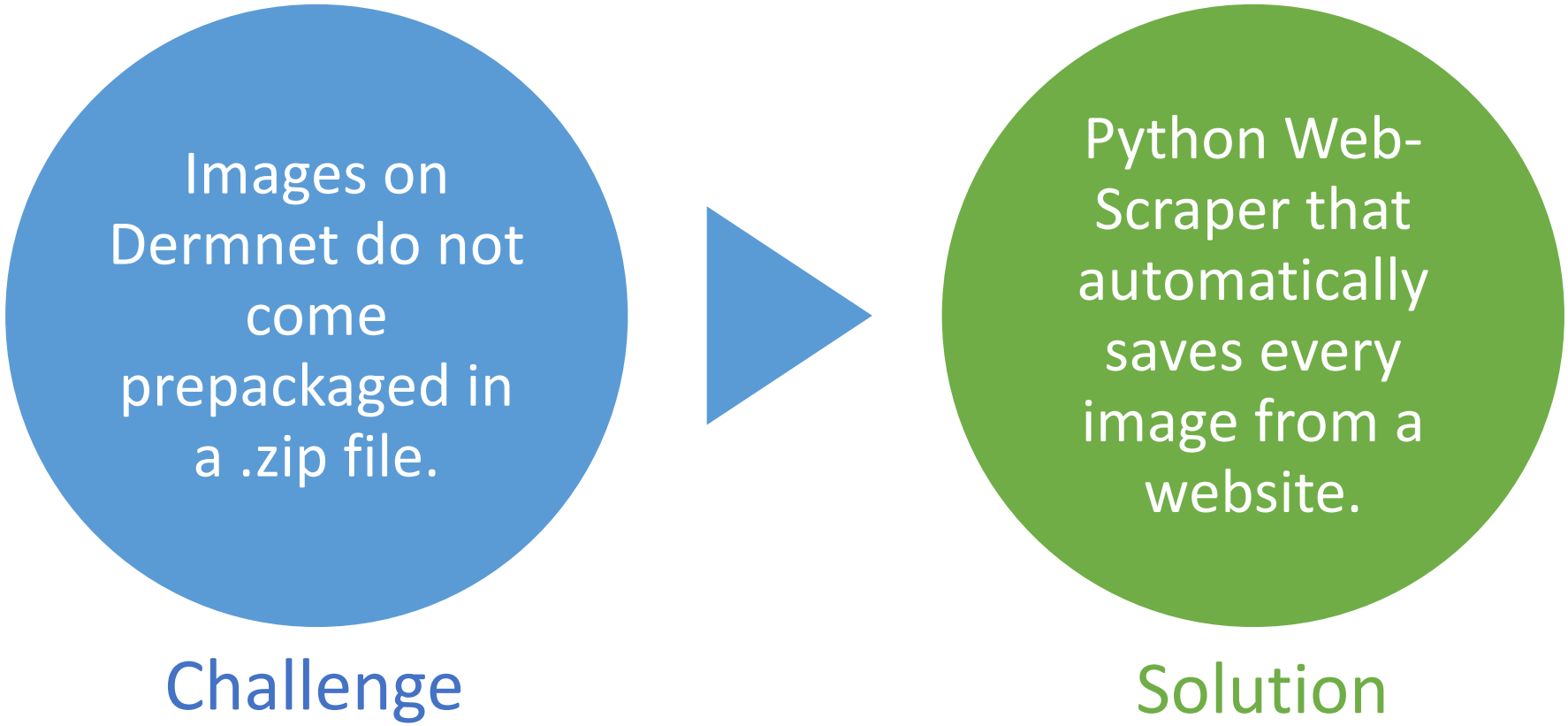
Dataset Overview

- Dermnet
  - Online photo dermatology source containing thousands of images of various skin conditions.

- Kaggle
  - “DermMel” dataset containing images of Melanoma and NotMelanoma. 8,903 pictures of Melanoma extracted.
  - “Derma Diseases” dataset containing 419 images of Melanoma and 413 images of Seborrheic Keratosis.



H


Images on Dermnet do not come prepackaged in a .zip file.
Python Web-Scraper that automatically saves every image from a website.
Challenge
Solution

Web Scraper Python Code

```python
def genClassImages(class_url):
    """Fetch list of class images
    @arg class_url: web url
    @returns class_images: list of images
    """
    images = []
    urls = genClassCategories(class_url)
    print('- Found {} total sub-classes for class.'.format(len(urls)))

    for i, url in enumerate(urls):
        print('-- Fetching images from sub-class [{}/{}]'.format(i + 1, len(urls)))
        images.extend(genCategoryImages(url))

    return images
```

```python
def genPageImages(url, image_list):
    """Finds all image links in a webpage and adds them to the image list.

    @arg url: web url; str
    @arg image_list: a list of image urls.
                     this will be modified in place.
    @return None
    """
    soup = soupify(url)
    thumbnails = soup.find_all("div","thumbnails")
    if thumbnails: ## there are thumbnails actually on the page
        for thumb in thumbnails:
            thumb_link = thumb.img['src']
            #use full image link instead of thumbnail link
            image_link = re.sub(r'Thumb',"",thumb_link)
            image_list.append(image_link)
```

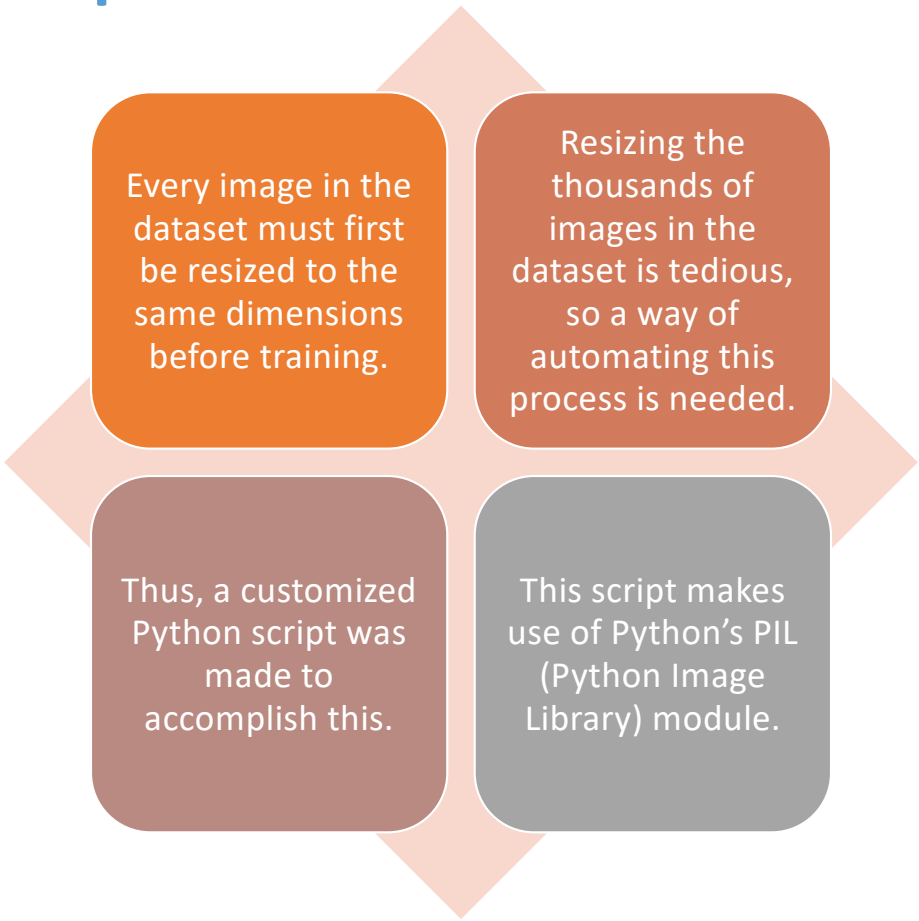
```python
if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('out_folder', type=str, help='where to store scraped images.')
    parser.add_argument('--dictionary', type=str, help='class2url dictionary path')
    args = parser.parse_args()

    print('Scraping DermNet for URLs.')
    if args.dictionary:
        with open(args.dictionary, 'rb') as fp:
            image_dict = pickle.load(fp)
    else:
        image_dict = genClass2URL()
    print(image_dict)

    n_images = 0
    for klass, images in image_dict.iteritems():
        n_images += len(images)

    n_downloaded = 0

    with open(os.path.join(args.out_folder, 'backup.pkl'),'wb') as fp:
        pickle.dump(image_dict, fp)
    print('Dumped dictionary of URLs to current directory.')

    # we will now download each image
    for klass, images in image_dict.iteritems():
        # create class folders, if it doesn't exist
        class_path = os.path.join(args.out_folder, klass)
        if not os.path.exists(class_path):
            os.mkdir(class_path)

        for image in images:
            image_name = os.path.basename(image)
            file_name = os.path.join(class_path, image_name)
            # download image
            try:
                f = urllib.urlopen(image).read()
                open(file_name, 'wb').write(f)
                n_downloaded += 1
                print('Downloaded [{}/{}] images.'.format(n_downloaded, n_images))
            except urllib.HTTPError:
                continue
```

s

Dataset Preparation

S

Resizing Script



```
from PIL import Image
import os, sys

path = "/Users/siddharthrajan/Users/siddharthrajan/dataset/validation/wart_viral_infections/"
dirs = os.listdir( path )

def resize():
    print("Resizing...")
    for item in dirs:
        if os.path.isfile(path+item):
            print("Processing: " + item)
            im = Image.open(path+item)
            imResize = im.resize((244,244), Image.ANTIALIAS)
            imResize.save("/Users/siddharthrajan/dataset/validation/wart_viral_infections/" + item, 'JPEG', quality=90)
    print("Done.")

resize()
```

Resizing Script Code

S

Final Dataset Size (60-20-20 split)

	Melanoma	Acne	Eczema	Seborrheic Keratosis	Warts	Scabies	Nail Fungus	Psoriasis	Tinea
Train	7182	527	929	1031	816	325	783	1056	977
Validation	2395	176	310	344	273	109	262	353	326
Test	2395	177	308	342	272	108	259	351	325
Total	11972	880	1547	1717	1361	542	1304	1760	1628

H

Generating an `.lst` file

- A .lst file is a tab-separated file with three columns that contains a list of image files. This is essentially a metadata file. **Required by our machine learning model implementation.**
  - First Column = Image Index (has to be unique for all images)
  - Second Column = Class Label Index (eg. Class 0 = acne, class 1 = eczema …)
  - Third Column = Relative Image Path

```
2623    2.000000    melanoma_nevi_moles/AUGmented_0_9668.jpeg
4338    8.000000    wart_viral_infections/warts-30.jpg
1473    2.000000    melanoma_nevi_moles/AUGmented_0_4505.jpeg
2808    2.000000    melanoma_nevi_moles/congenital-nevus-26.jpg
36  0.000000    acne_and_rosacea/acne-open-comedo-59.jpg
1848    2.000000    melanoma_nevi_moles/AUGmented_0_6291.jpeg
865 2.000000    melanoma_nevi_moles/AUGmented_0_1782.jpeg
362 1.000000    eczema/lichen-simplex-chronicus-10.jpg
1435    2.000000    melanoma_nevi_moles/AUGmented_0_4354.jpeg
```

.lst file contents

```
acne_and_rosacea 0
eczema 1
melanoma_nevi_moles 2
nail_fungus 3
psoriasis 4
scabies 5
seborrheic_keratoses 6
tinea_ringworm 7
wart_viral_infections 8
```

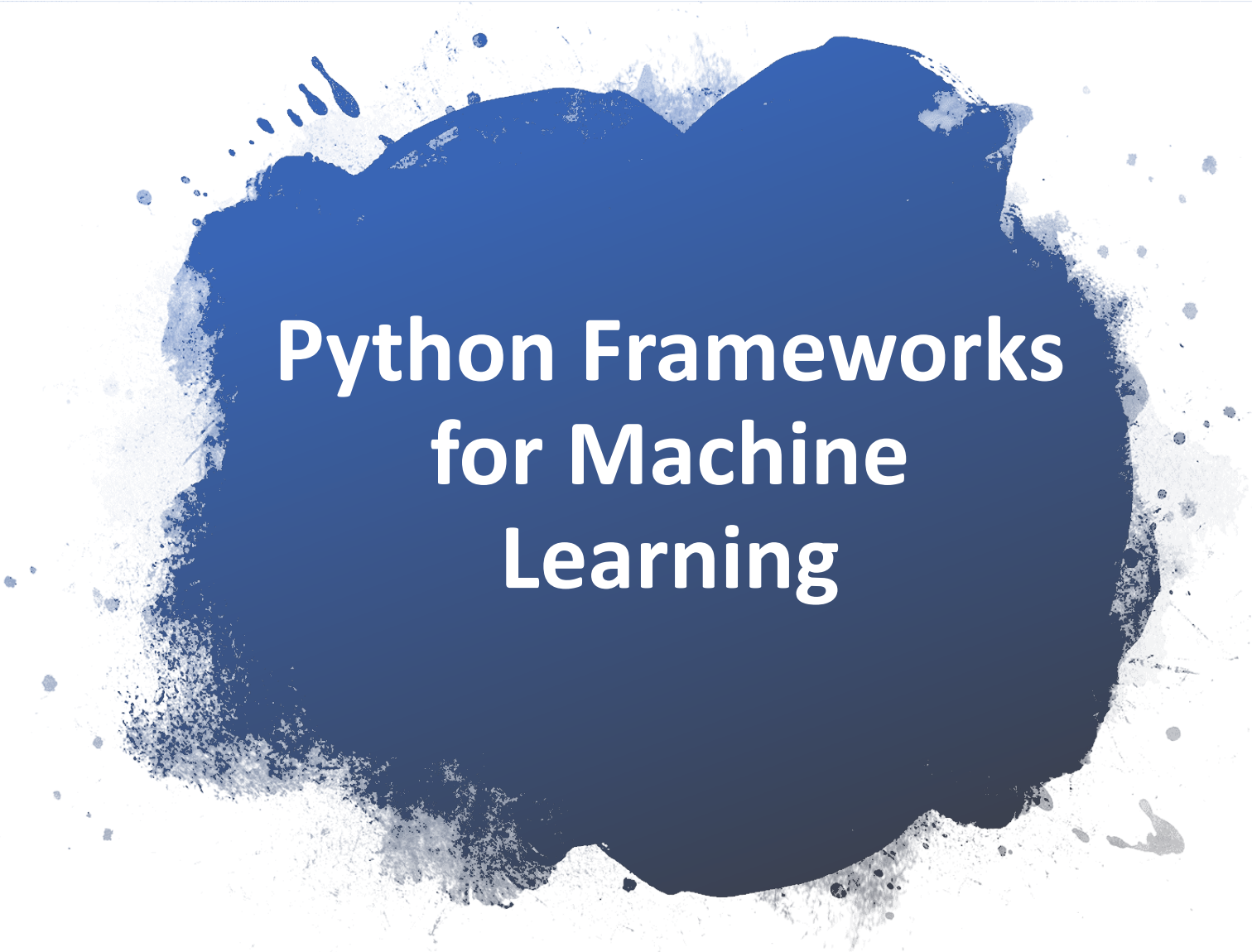
All Class Labels (ground truth)

H

im2rec.py

```
(tensorflow) C:\Users\Naman\Documents\senior-design-research>python im2rec.py --list --recursive validation senior-design-dataset-new/validation/
acne_and_rosacea 0
eczema 1
melanoma_nevi_moles 2
nail_fungus 3
psoriasis 4
scabies 5
seborrheic_keratoses 6
tinea_ringworm 7
wart_viral_infections 8
```

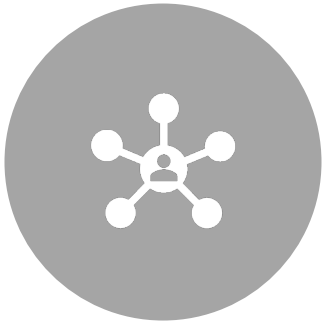
F



F

Common Python Libraries For AI/ML

TENSORFLOW – LIBRARY DEVELOPED BY GOOGLE THAT USES DATA FLOW GRAPHS WHERE EACH NODE IN THE GRAPH REPRESENTS MATHEMATICAL OPERATION AND EACH EDGE BETWEEN NODES IS A TENSOR.



PYTORCH – MOST NOTABLE FEATURE INCLUDE COMPUTATIONS WITH STRONG GPU ACCELERATION SUPPORT AND BUILDING DEEP NEURAL NETWORKS. AVOIDS STATIC GRAPHS, ALLOWING DEVELOPERS AND RESEARCHERS TO CHANGE HOW THE NETWORK BEHAVES INSTANTANEOUSLY

KERAS - OFFERS THE ADVANTAGES OF BROAD ADOPTION, SUPPORT FOR A WIDE RANGE OF PRODUCTION DEPLOYMENT OPTIONS, INTEGRATION WITH AT LEAST FIVE BACK-END ENGINES (TENSORFLOW, CNTK, THEANO, MXNET, AND PLAIDML), AND STRONG SUPPORT FOR MULTIPLE GPUS AND DISTRIBUTED TRAINING.

N

Cloud Computing Platforms

N

Amazon EC2 (Elastic Compute Cloud)

Fully Customizable, Cloud Computing Solution offered to developer

Can be thought of as a virtual “computer” on the cloud with fully customizable hardware ranging including:

1. RAM
2. Architecture
3. Storage
4. Network Performance

Developers are given **full** control over said “computers”

Specification	Value
Architecture	x86_64
Memory	64GB
Storage	Customizable
Network Performance	High
vCPUs	4

N

Drawbacks

EC2s are extremely customizable, which also a disadvantage

EC2s required a lot of manual work including but not limited to

1. Installing all project dependencies
2. Establishing scripts to train model in background
3. Creating custom HTTP endpoints to bridge gap between our User Interface and Machine Learning Model

EC2 is not a service that is **dedicated** to machine learning.

N

Amazon SageMaker

A **dedicated** cloud service for developing, training, and deploying Machine Learning Models.

Considers the **entire** development journey, and includes such sub-services as

- Notebook Instances
- Managed Training Jobs
- Managed Hyperparameter Tuning Jobs
- Managed Endpoints for hosting trained models on the cloud

F

SageMaker Development Journey

Initializing a Training Image

Providing a Dataset

Configuring Hyperparameters for Training Job

Deploying Training Job

Hosting Endpoint on Cloud

Initializing a Training Image

SageMaker provides pre-initialized, untrained Neural Nets in docker files named *training images.*

- These are an alternative to writing `keras` or `tensorflow` code and are provided for various use cases.

The team used the `image-classification` training image.

- A docker image containing Convolutional Neural Network architecture specifically for the purpose of Image Classification.

```
%%time
import boto3
import re
from sagemaker import get_execution_role
from sagemaker.amazon.amazon_estimator import get_image_uri

role = get_execution_role()
print("Role = " + role)

bucket='senior-design-app-bucket' # customize to your bucket

training_image = get_image_uri(boto3.Session().region_name, 'image-classification')
```

Providing a Dataset

Prior to training, provide S3 (Amazon Simple Storage Service) URI to model using the prefix `s3://{bucket_name}`

S3 buckets use regular Linux style file systems and can hold massive amounts of data at once. The structure of our dataset storage solution was as follows.

`train/`
`validation/`
`test/`
`train_lst/`
Folder containing `train.lst` file, a metadata list corresponding each image path with its ground truth class label
`validation_lst/`
Folder containing `validation.lst` file

```
# configuring the s3 uris for input data
s3train = 's3://{}/updated_dataset/train/'.format(bucket)
s3validation = 's3://{}/updated_dataset/validation/'.format(bucket)
s3train_lst = 's3://{}/updated_dataset/train_lst/'.format(bucket)
s3validation_lst = 's3://{}/updated_dataset/validation_lst/'.format(bucket)
```

S

Configuring Hyperparameters for Training Job

Configuring hyperparameters for CNN, including but not limited to the parameters listed below.

- **num_layers** *(number of layers)*
- **image_shape** *(dimensions of each image in the dataset)*
- **num_training_samples** *(number of images in the training split)*
- **num_classes** *(number of defined classes in the dataset)*
- **mini_batch_size**
- **epochs**
- **learning_rate**

```
# The algorithm supports multiple network depth (number of layers). They are
# we'll use 50 layers
num_layers = "101"
# we need to specify the input image shape for the training data
image_shape = "3,244,244"
# we also need to zpecify the number of training samples in the training set
# for our dataset this is 8522 (skin dataset)
num_training_samples = "14707"
# specify the number of output classes
num_classes = "9"
# batch size for training
mini_batch_size =  "32"
# number of epochs
epochs = "100"
# learning rate
learning_rate = "0.01"
```

S

Deploying Training Job

Training jobs can be deployed once a model's hyperparameters and dataset resources are defined.

> In laymen terms, this means that the model can be **trained** on the cloud.

Before doing this, the `training_params` variable must be set, defining the following important specifications.

- **"AlgorithmSpecification"**
- **"ResourceConfig"** (defining the type of compute power is desired to train the model)
- **"HyperParameters"** (defining the hyper parameters)
- **"InputDataConfig"** (providing information on how to retrieve the training and validation datasets)

```
training_params = \
{
    # specify the training docker image
    "AlgorithmSpecification": {
        "TrainingImage": training_image,
        "TrainingInputMode": "File"
    },
    "RoleArn": role,
    "OutputDataConfig": {
        "S3OutputPath": 's3://{}/{}/output'.format(bucket, job_name_prefix)
    },
    "ResourceConfig": {
        "InstanceCount": 1,
        "InstanceType": "ml.p2.xlarge",
        "VolumeSizeInGB": 50
    },
    "TrainingJobName": job_name,
    "HyperParameters": {
        "image_shape": image_shape,
        "num_layers": str(num_layers),
        "num_training_samples": str(num_training_samples),
        "num_classes": str(num_classes),
        "mini_batch_size": str(mini_batch_size),
        "epochs": str(epochs),
        "learning_rate": str(learning_rate)
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 360000
    },
```

Hosting Endpoint on Cloud (real-time inference)

Important Feature allowing trained model to be hosted on the cloud

- Accessed using HTTP (hypertext transfer protocol)

`endpoint_config` variable created, defining type of cloud instance hosting endpoint, and model to be hosted

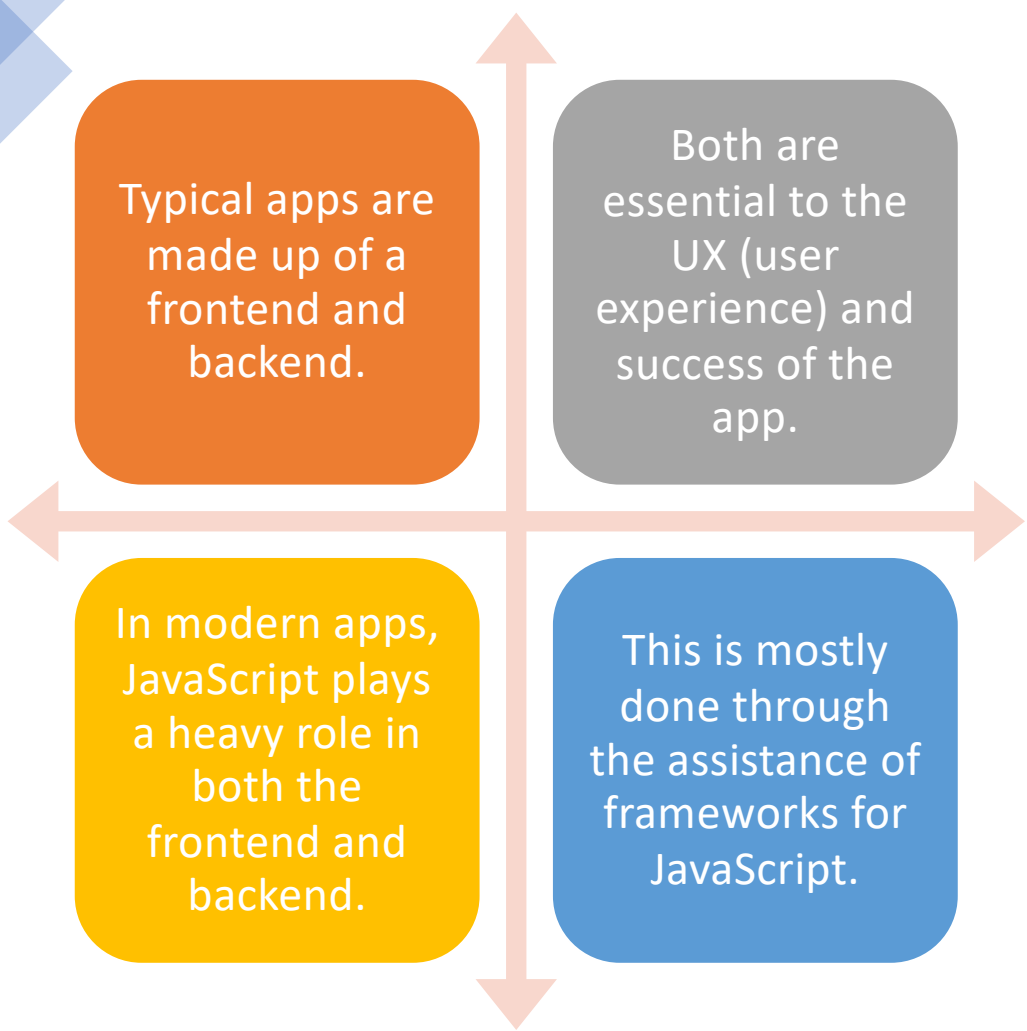
- `"InstanceType"` and `"ModelName"`, respectively

```
endpoint_config_name = "skinnefy-training-version2-endpoint-config"
endpoint_config_response = sage.create_endpoint_config(
    EndpointConfigName = endpoint_config_name,
    ProductionVariants=[{
        'InstanceType':'ml.m4.xlarge',
        'InitialInstanceCount':1,
        'ModelName':model_name,
        'VariantName':'AllTraffic'}])

print('Endpoint configuration name: {}'.format(endpoint_config_name))
print('Endpoint configuration arn:  {}'.format(endpoint_config_response['EndpointConfigArn']))
```

s

Web Frameworks

S



UI Solutions

F

Different JS Frameworks

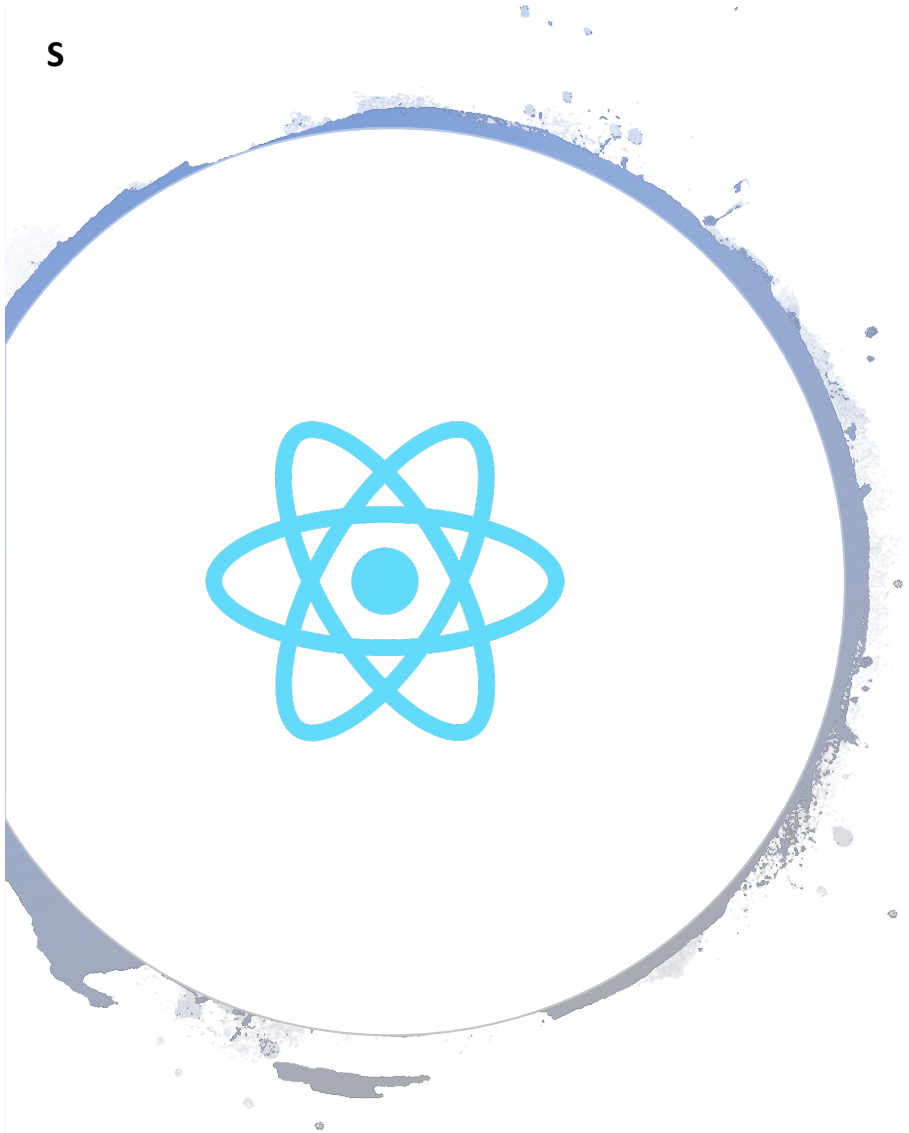
- The purpose of JavaScript frameworks is to both enhance and ease the experience of building common application components.
- These include things such as search bars, login menus, buttons, etc.
- The most popular frameworks in the industry today are React, Angular, Vue, Node, Express, and many more.

s

A Breakdown of Angular.js

- Angular is a popular frontend JS framework.
- Angular gives the ability to extend HTML so that HTML pages behave dynamically.
- This is done by implementing custom HTML tags that define in detail all the behaviors.
- There are only a few calls to backend JavaScript functions.

s



A Breakdown of React.js

- React is another, and by far the most popular, JS framework for building functional user interfaces.
- Unlike Angular, React makes little to no use of HTML, and instead all the implementation is done with JavaScript.
- React makes use of functional and class components in order to implement items seen on an app screen.

The Final Verdict on Frameworks

- The decision was made to use a React environment for development of the app.
- This decision was mostly made due to prior experience and interest.
- However, to develop an app specifically for iOS/Android, a slight extension to the React environment is needed.
- Thus, the final decision was to use **React Native** (explained later).

N

Mobile Application Frameworks

N

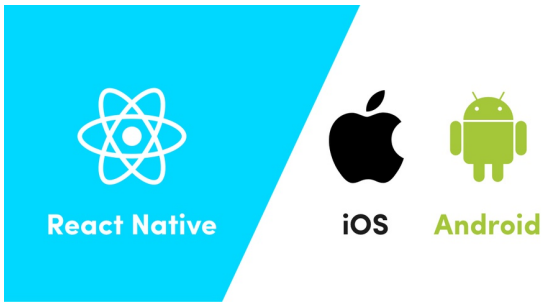
Front-end Solutions

N

React Native

Offers a JavaScript-based framework for building smartphone applications

- Built on the React environment
- Allows developers to produce applications compatible with both iOS and Android



Advantages	Disadvantages
Source code is the **same** for both versions of the application (iOS and Android)	Not as optimized (relatively), leading to **marginally** slower code and inefficient applications if built for scale
Code more re-usable and much easier to debug at once.	React Native APIs do not cover all the features and access that native coding can provide to developers
Less “expertise”/workers require.	

We **chose** React Native as our application-building framework, due to previous experience

Setting up our React Native Project

Initializing an *Expo* project

A tool-chain around bare-bones React Native, enhancing developmental experience with features such as

- Cross collaboration between different devices
- Fast-refresh to allow for real time development
- Cloud based code sharing through Expo Snacks

```
npm install expo-cli --global
```

```
expo start
```

Installing Necessary Third-Party Packages

Packages enhance application by providing reliable functionality to application.

- No need to reinvent the wheel.

Name of Package	Description
`react-navigation`	Enables navigation (clicking tabs, swiping, going back etc.)
`react-native-elements`	Set of polished, appealing UI components (buttons, text fields, avatars, etc.)
`firebase`	Gives access to Firebase, Google's back-end service managing our database, authentication, and storage
`aws-sdk`	Gives access to Amazon SageMaker endpoint hosting our model, through the application. Allows inference.

H

H

Google Firebase

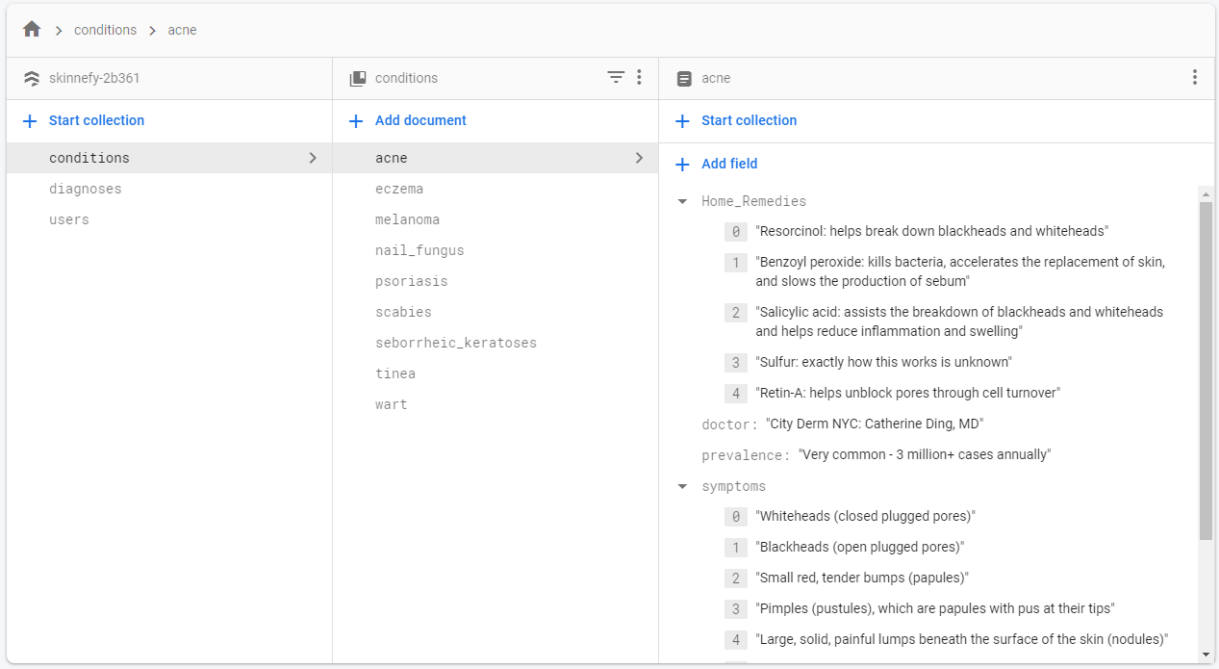
Real-time Database (Firestore)

User Authentication

Storage

H

Firebase Database



collections

documents

data contained **in** document

Real-time NoSQL database

> **Documents** are unstructured data analogous to rows in SQL database
>
> **Collections** are multiple documents stored based on a common category

Populated the database with the relevant information to be used on the remedies tab, along with user profile information.

Fields that have > 1 elements are stored as arrays to organize and display text on the UI neatly.

H

Code for Database

Initializing Instance of `firestore` (real-time database)

```
var db = firebase.firestore();
```

Pulling document data once

- Querying information stored in a document just once
- Does not establish a listener or a subscription. If further changes are made to document, they do not update real time on the application.
- Use cases → data stored in remedy screen (not expected to frequently change)

```
export const useDiagnosisOnce = (postDocName) => {
    const [docState, setDocState] = React.useState({ isLoading: true, data: null });
    const compileData = async () => {
        var diagnosisDoc = await db.collection("diagnoses").doc(postDocName).get();
        setDocState({
            isLoading: false,
            data: diagnosisDoc.data(),
        });
    }
    React.useEffect(() => {
        compileData();
    }, []);
    return docState;
}
```

Subscribing to a Document in a collection

- "subscribing" → attaching a listener to document and expecting real-time updates if document state is changed
- Use cases: displaying profiles (ever changing states – number of diagnoses taken constantly changing)

```
export const useFirestoreDoc = (collectionName, docName) => {
    const [docState, setDocState] = React.useState({ isLoading: true, data: null });
    React.useEffect(() => {
        return db.collection(collectionName)
            .doc(docName)
            .onSnapshot(doc => {
            setDocState({ isLoading: false, data: doc.data() });
        });
    }, []);
    return docState;
}
```

H

Firebase User Authentication

- Firebase Authentication provides easy-to-use SDKs, and ready-made UI libraries to authenticate users to applications.

- Has built in email/password authentication system and supports OAuth2 for Google, Facebook, Twitter and GitHub.

H

Code for User Authentication

Initializing Instance of auth (authentication service)

```
var auth = firebase.auth();
```

Setting up Authentication Listener

- Continually checks user status
- On status change (signed in → out, or vice-versa) show/hide application
- Logged in → show application. Logged out → show login screen

```
export const useAuth = () => {
    const [state, setState] = React.useState(() => {
        const user = firebase.auth().currentUser;
        return { initializing: !user, user, }
    })
    function onChange(user) {
        setState({ initializing: false, user });
    }
    React.useEffect(() => {
        // listen for auth changes
        const unsubscribe = firebase.auth().onAuthStateChanged(onChange);
        // unsubscribe to the listener when unmounting
        return () => unsubscribe();
    }, []);
    return state;
}
```

Functionality for Logging out

- Called only when the logout button is pressed in the application

```
export function logout() {
    auth.signOut().then(() => console.log("User Signed out"))
    .catch(() => console.log(error.message));
}
```

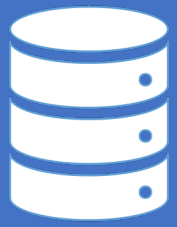
Functionality for Logging in

- Called only when the "sign-in" button is pressed in the application

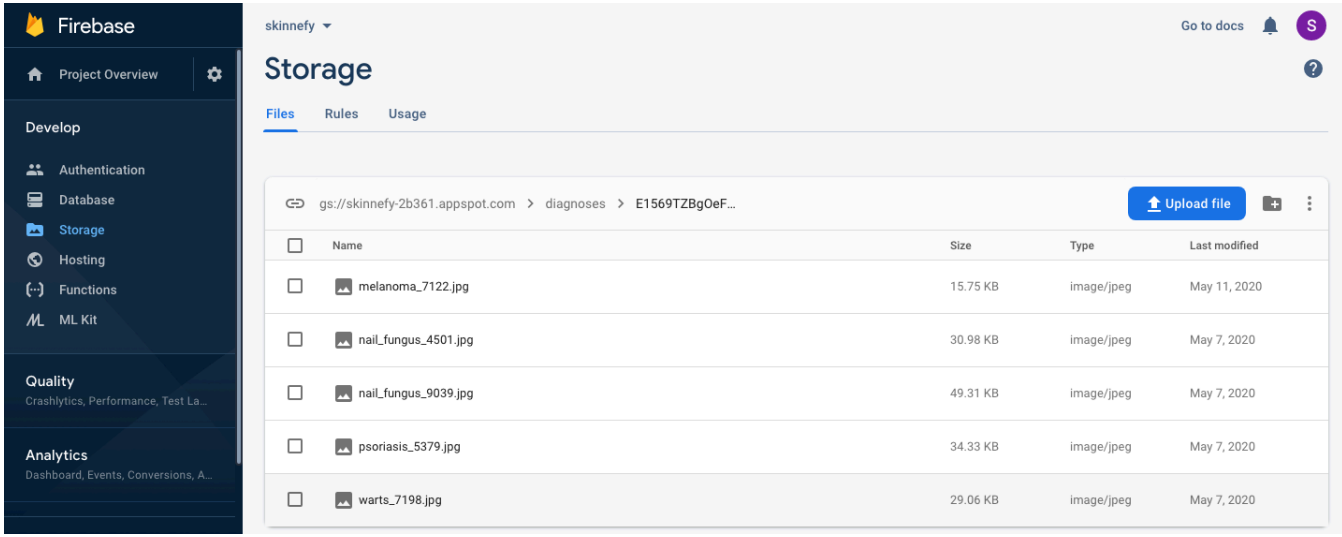
```
export function loginWithPassword(email, password) {
    return auth.signInWithEmailAndPassword(email, password);
}
```

H

Firebase Cloud Storage



- Cloud Storage stores files in a Google Cloud Storage bucket, making them accessible through Firebase.
- Store images, audio, video, or other user-generated content.
- Saving images previously taken and displaying on user profile



Cloud for Firebase Cloud Storage

Initializing Instance of `storage` (storage service)

```
var storage = firebase.storage();
```

Storing User Diagnosis Images (camera pictures)

- Convert to Blob (Binary Large Object)
- Upload to Firebase Storage using `.put` method
- Naming of images made unique to avoid conflicts
- Upload status is monitored (0-100%)
- After upload, record of diagnosis image is added to user's profile

```
export async function uploadDiagnosisSequence(user, payload, image) {
    /* require the user so that the image is stored correctly in
    the storage bucket. the title is used to name the file, while image
    is a json with a key for the base64 encoded string */
    let suffix = Math.round(Math.random() * 10000);
    const path = "diagnoses/" + user + "/" + payload.title.toLowerCase().replace(/ /g, "_") + "_" + suffix.toString() + ".jpg";
    var storageRef = storage.ref();

    const response = await fetch(image.uri);
    const blob = await response.blob();

    var uploadTask = storageRef.child(path).put(blob);
    uploadTask.on(firebase.storage.TaskEvent.STATE_CHANGED, // or 'state_changed'
        function(snapshot) {
            // Get task progress, including the number of bytes uploaded and the total number of bytes to be uploaded
            var progress = (snapshot.bytesTransferred / snapshot.totalBytes) * 100;
            console.log('Upload is ' + progress + '% done');
            switch (snapshot.state) {
                case firebase.storage.TaskState.PAUSED: // or 'paused'
                    console.log('Upload is paused');
                    break;
                case firebase.storage.TaskState.RUNNING: // or 'running'
                    console.log('Upload is running');
                    break;
            }
        }, function(error) {
            // A full list of error codes is available at
            // https://firebase.google.com/docs/storage/web/handle-errors
            console.log(error.code);
        }, saveToDatabase
    );
```

Amazon Web Services JavaScript SDK

Gave access to SageMaker endpoint

- This allowed us to get inferences from our trained models on the cloud, from the React Native Application

Compatible with React Native application (React Native based on JavaScript)

`invokeEndpoint()` method allows developers to specify an endpoint name, prediction data, and content type

- Returns inference in array format

```
var params = {
  Body: Buffer.from('...') || 'STRING_VALUE' /* Strings will be
  EndpointName: 'STRING_VALUE', /* required */
  Accept: 'STRING_VALUE',
  ContentType: 'STRING_VALUE',
  CustomAttributes: 'STRING_VALUE',
  TargetModel: 'STRING_VALUE'
};
sagemakerruntime.invokeEndpoint(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else     console.log(data);           // successful response
});
```

```
export function useEndpoint(image) {
    const [queryState, setQueryState] = React.useState({ isLoading: true, data: null, err: nu

    const sendAndRetrieve = async () => {
        try {
            const response = await fetch(image.uri);
            const blob = await response.blob();
            sagemakerruntime.invokeEndpoint({
                Body: blob,
                EndpointName: endpoint_name,
                ContentType: "application/x-image",
            }, function(err, data) {
                if(err) {
                    setQueryState({
                        isLoading: false,
                        data: null,
                        err: err.message
                    })
                }
                else {
                    let prediction = prepareResponse(data.Body);
                    setQueryState({
                        isLoading: false,
                        data: prediction,
                        err: null
                    })
                }
            });
        }
        catch (error) {
            console.log(error)
        }
    }
    React.useEffect(() => {
        sendAndRetrieve();
    }, []);
```

S

Results

s

Machine Learning Model

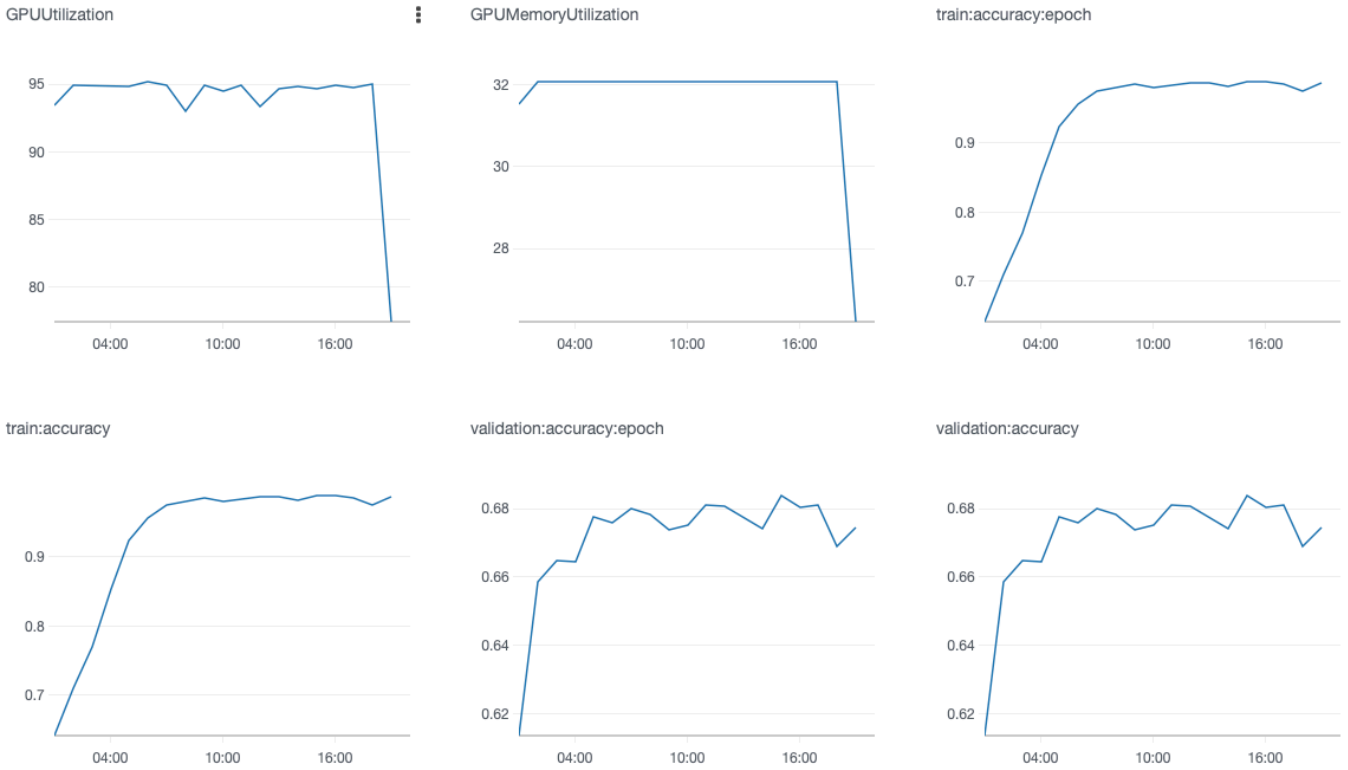
S

Training Results

Training Job Name	Epochs	# Classes	Learning Rate	Mini Batch Size	# Layers	Training Accuracy	Validation Accuracy
skinclassification-test1-2020-03-18-02-02-53	30	3	0.01	64	50	0.991	0.900
skin-condition-tuning-job-4-022-2ec93c08	30	3	0.02	64	50	0.996	0.900
skinnefy-model-tuning-job-v5-010-bf49c55a	30	10	0.001	32	50	0.957	0.665
skinnefy-training-version1	75	10	0.01	32	110	0.795	0.643
skinnefy-training-version2	75	10	0.01	32	110	0.983	0.639
skinnefy-training-job-version-3	100	9	0.01	16	101	0.986	0.669
skinnefy-training-job-version-4	100	9	0.01	8	101	0.974	0.753

s

Training Visualized



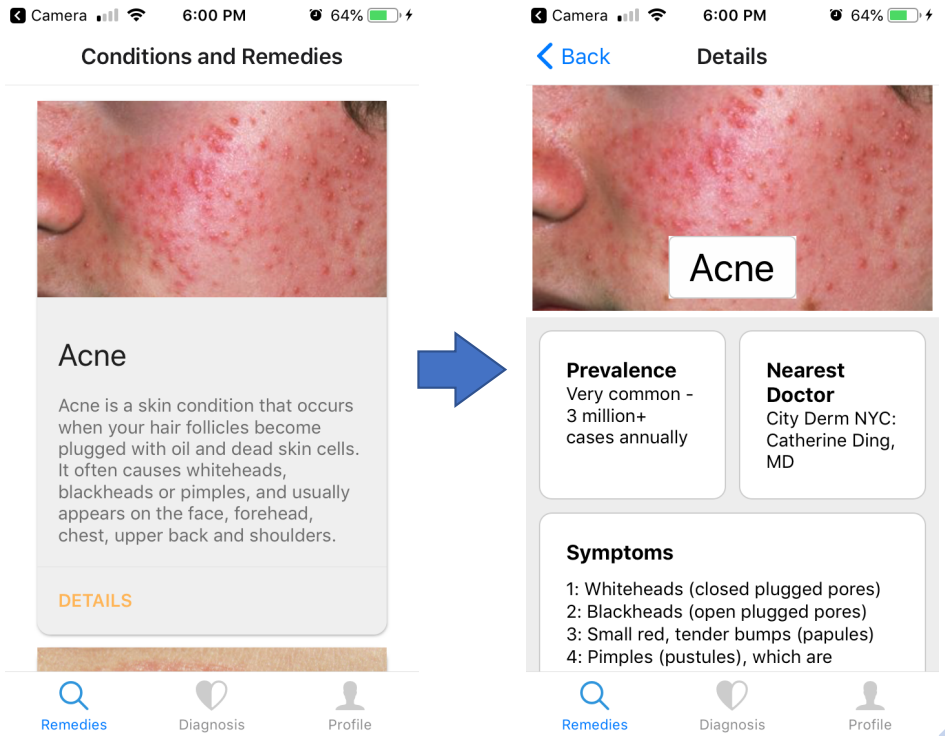
s

User Interface

S

UI – Remedy Tab

- The remedy tab is made with a card design layout.
- Upon pressing one of the cards, the user is taken to a second screen to see the details of the condition listed on the card.



```
render() {
    return (
        <React.Fragment>
        <ScrollView style={{ flex: 1 }} contentContainerStyle={styles.container}>
            {prepared_conditions.map((condition, key) => {
                return (
                    <Card style={{
                        backgroundColor: "#efefef",
                        borderRadius: 10, elevation: 2,
                    }}
                    key={key}>
                        <CardImage
                            source={{uri : condition.image}}
                            style={{
                                height: 150,
                            }}
                        />
                        <CardTitle title={condition.name} />
                        <CardContent text={condition.description} />
                        <CardAction separator={true} inColumn={false}>
                            <CardButton
                                onPress={() => {this.props.navigation.navigate('Details', {
                                    query: condition.ref,
                                    displayTop: condition.name,
                                    description: condition.description,
                                    image: condition.image,
                                })}}
                                title="Details"
                                color="#FEB557"
                            />
                        </CardAction>
                    </Card>
                )
            })
```

```
export default class DetailsScreen extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            isLoading: true,
            data: null,
        }
    }
    componentDidMount() {
        db.collection('conditions').doc(this.props.route.params.query).get().then(doc => {
            this.setState({
                isLoading: false,
                data: doc.data(),
            });
        })
    }

    render() {
        const { image, displayTop, description } = this.props.route.params;
        return (
            <React.Fragment>
            { this.state.isLoading && <Loading/>}
            { !this.state.isLoading &&
            <ScrollView style={{ flex: 1 }} contentContainerStyle={styles.container}>
                <View style={{
                    padding: 5,
                    backgroundColor: "#fff",
                    elevation: 5,
                }}>
                    <ImageBackground
                        source={{ uri: image }}
                        style={{
                            height: 200,
                            justifyContent: "flex-end",
                        }}
                    >
                        <View style={{
                                flexDirection: "row",
                                flexWrap: "nowrap",
                                justifyContent: "center"
                            }}
                        >
                            <Text style={styles.title}>
                                {displayTop}
                            </Text>
                        </View>
                    </ImageBackground>
                </View>
                <View style={{
                        paddingVertical: 10,
                    }}
                >
                    <View style={{
                            flexDirection: "row",
                            justifyContent: "space-between",
                            marginHorizontal: 10, marginBottom: 10,
                        }}
                    >
                        <View style={{
                                flex: 0.5,
                                padding: 20,
                                backgroundColor: "white",
                                borderRadius: 10, borderColor: "#ccc",
                                borderWidth: 1, marginRight: 5,
```

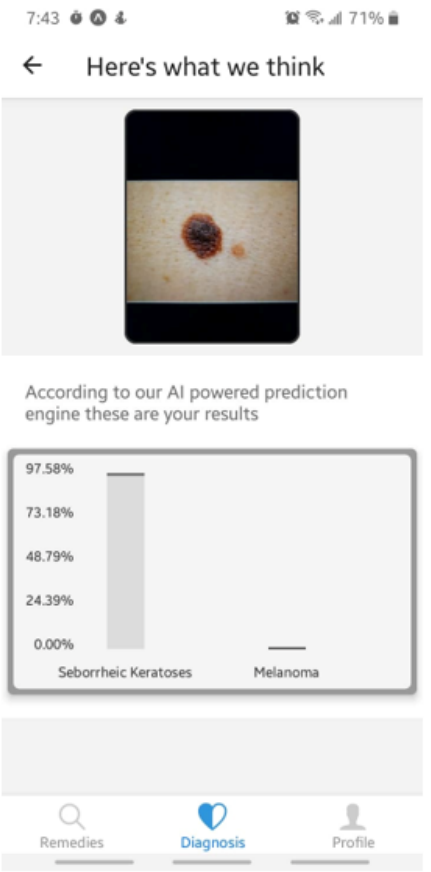
Diagnosis Tab

Allows users to take real-time images using their cameras and receive a skin-condition inference

- Inference presented as a bar graph showing top two results (percentage confidence values)

Diagnosis Tab comprised of two distinct screens

1. Camera Screen
   - Uses the `<Camera>` React Native API to get data from smartphone's camera hardware
   - Image sent to SageMaker endpoint for inference
2. Results Screen
   - Displays preview of skin condition image taken
   - Followed by inference



N

```
<Camera style={{ flex: 1, }} type={type} ratio={CAMERA_RATIO}
  autoFocus={false} pictureSize={size}
  ref={ref => { camera = ref; }} autoFocus
  onCameraReady={getPictureSizes}
>
  <View style={{flex: 1, flexDirection: "column", justifyContent: "flex-end"}}>
    <Text style={{fontWeight: "bold", padding: 25, fontSize: 15, color: "#fff", alignSelf: "center",}}>
      Take a picture of your skin condition
    </Text>
    <View style={{paddingBottom: 20, flexDirection: "row", justifyContent: "space-evenly",
        alignItems: "center",}}>
        <TouchableOpacity
            onPress={() => {
            camera.takePictureAsync({
              base64: true,
              compression: 0.0,
              onPictureSaved: onPictureSaved});
            }}
        >
            <Ionicons
              name='ios-radio-button-off'
              color="#fefefe"
              size={70}
            />
        </TouchableOpacity>
        <TouchableOpacity
            onPress={() => {
            setType(
            type === Camera.Constants.Type.back
                ? Camera.Constants.Type.front
                : Camera.Constants.Type.back
            );
        }}>
            <Ionicons
            name='ios-refresh'
            color="#fefefe"
            size={35}
            />
        </TouchableOpacity>
    </View>
  </View>
</Camera>
```

```
const onPictureSaved = photo => {
  navigation.navigate('Results', { data: photo })
}
```

```
<BarChart
    // style={graphStyle}
    fromZero={true}
    segments={4}
    withInnerLines={false}
    data={{
        labels: data.labels.slice(0, 2),
        datasets: [
            {
                data: data.dataset.slice(0, 2).map(val => {
                    return val * 100;
                }),
            },
        ],
    }}
    yAxisSuffix="%"
    width={Dimensions.get('window').width - 20}
    height={200}
    chartConfig={{
        backgroundColor: '#1cc910',
        backgroundGradientFrom: '#fff',
        backgroundGradientTo: '#999',
        decimalPlaces: 2, // optional, defaults to 2dp
        color: (opacity = 255) => `rgba(0, 0, 0, ${opacity})`,
        style: {
        borderRadius: 16
        },

    }}
    style={{
        borderRadius: 5,
        padding: 5,
        backgroundColor: "#999",
        elevation: 5,
    }}
/>
```

H

Profile Tab

Basic User Information including

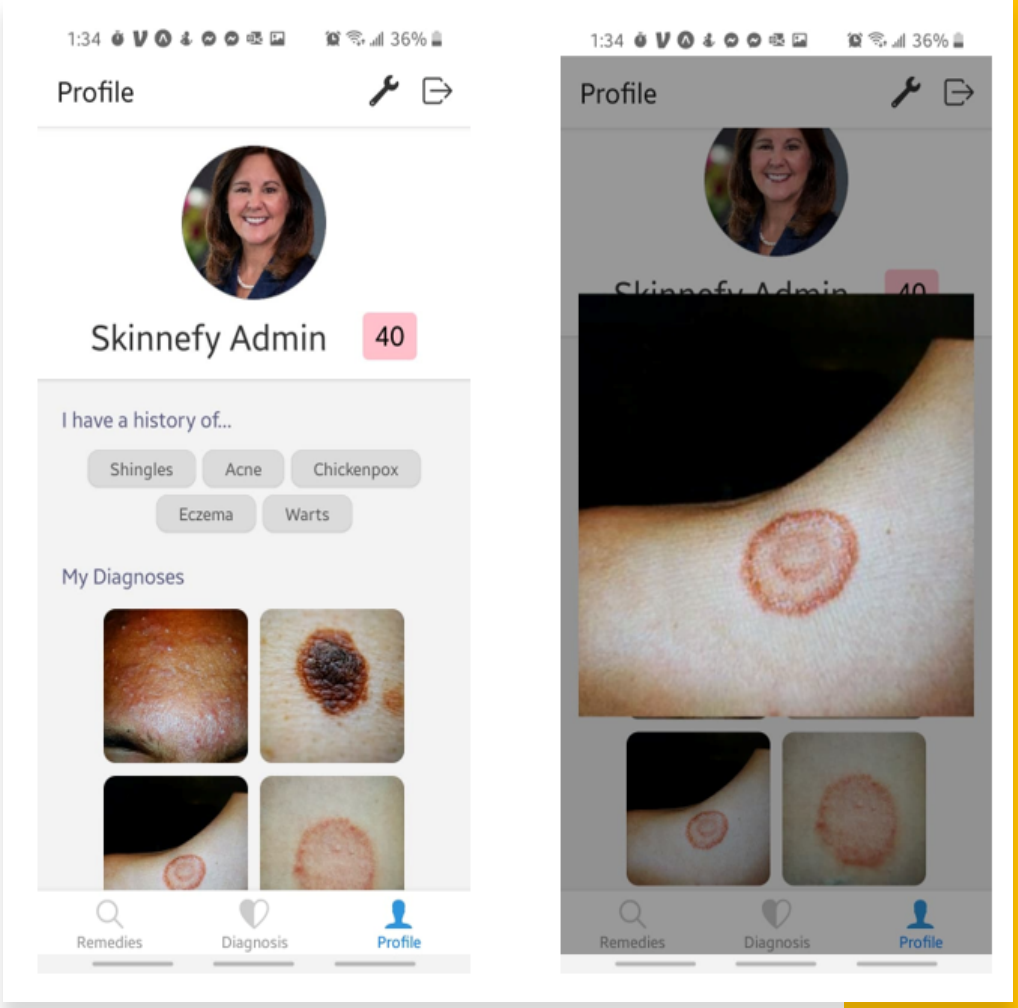
1. User Avatar
2. User's Full Name
3. Age and Sex (depicted by background color pink or blue)

History of Previous Skin Conditions

My diagnoses section displays the previous diagnoses you have given yourself using the ML algorithm.

Can prove useful when, say you are visiting a professional, and are able to show a history of your conditions, and what you think might be the problem

Overlay covering whole screen with diagnosis picture when pressed



N

```
<Text style={{fontSize:15, color: "#5a5d81"}}>
    I have a history of...
</Text>
<View style={{
    paddingVertical: 10,
    flexDirection: "row", flexWrap: "wrap",
    justifyContent: "center"
}}>
    {
        data.history.map((x, i) => {
            return (
                <Badge
                    badgeStyle={{
                    backgroundColor: "#dbdbdb",
                    borderColor: "#cccccc",
                    borderWidth: 0.5,
                    elevation: 1,
                    padding: 14,
                    margin: 3,
                    }}
                    key={i}
                    value={x}
                    textStyle={{
                        color: "#555"
                    }}
                />
            )
        })
    }
</View>
```

In conclusion…

As a team we were able to…

- Highlight our capabilities with modern-day technology
- Get valuable experience with regards to Machine Learning and App Development

However, we could have…

- Made our dataset more complete
- Built more upon our CNN's accuracy
- Explored and evaluated more cloud platforms

Nevertheless, we consider our efforts be a success, as we were able to manifest our original vision into reality

All

Thank you.