CLOUD-BASED SKIN LESION DIAGNOSIS SYSTEM USING CONVOLUTIONAL NEURAL NETWORKS

by

Esad Akar

A Thesis Submitted to the Faculty of

College of Engineering & Computer Science

In Partial Fulfillment of the Requirements for the Degree of

Master of Science

Florida Atlantic University

Boca Raton, FL

December 2018

CLOUD-BASED SKIN LESION DIAGNOSIS SYSTEM USING

CONVOLUTIONAL NEURAL NETWORKS

by

Esad Akar

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Borko Furht, Department of Computer & Electrical Engineering and Computer Science, and has been approved by the members of his supervisory committee. It was submitted to the faculty of the College of Engineering & Computer Science and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

SUPERVISORY COMMITTEE:

Borko Furht, Ph.D.
Thesis Advisor

Hari Kalva, Ph.D.

Nurgun Erdol, Ph.D.
Chair, Department of Computer & Electrical Engineering and Computer Science

Xingquan (Hill) Zhu, Ph.D.

Stella Batalama, Ph.D.
Dean, College of Engineering & Computer Science

Khaled Sobhan, Ph.D.
Interim Dean, Graduate College

November 28, 2018
Date

ACKNOWLEDGEMENTS

To my advisor, Borko Furht, Ph.D.: because this would not have been possible with your guidance and trust in me. Many Thanks!

I am grateful to my siblings, mother, and father who have provided me through moral and emotional support in my life.

With a special mention to Oge Marques, Ph.D. and Hari Kalva, Ph.D. who have been extremely generous in sharing their knowledge.

Thanks for all your encouragement!

ABSTRACT

Author: Esad Akar

Title: Cloud-based Skin Lesion Diagnosis System using Convolutional Neural Networks

Institution: Florida Atlantic University

Thesis Advisor: Dr. Borko Furht

Degree: Master of Science

Year: 2018

Skin cancer is a major medical problem. If not detected early enough, skin cancer like melanoma can turn fatal. As a result, early detection of skin cancer, like other types of cancer, is key for survival. In recent times, deep learning methods have been explored to create improved skin lesion diagnosis tools. In some cases, the accuracy of these methods has reached dermatologist level of accuracy. For this thesis, a full-fledged cloud-based diagnosis system powered by convolutional neural networks (CNNs) with near dermatologist level accuracy has been designed and implemented in part to increase early detection of skin cancer. A large range of client devices can connect to the system to upload digital lesion images and request diagnosis results from the diagnosis pipeline. The diagnosis is handled by a two-stage CNN pipeline hosted on a server where a preliminary CNN performs quality check on user requests, and a diagnosis CNN that outputs lesion predictions.

CLOUD-BASED SKIN LESION DIAGNOSIS SYSTEM USING CONVOLUTIONAL NEURAL NETWORKS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

OVERVIEW

1.1 INTRODUCTION

Development of artificial intelligence may have significant impact in the future in important fields like medicine and help to provide more efficient and affordable services to patients around the world. Intelligent diagnosis systems based on artificial intelligence can aid doctors with tasks like disease diagnosis. Recently, deep learning methods have been explored to provide solutions to image-based diagnosis tasks such as skin lesion and cancer classification. Improved diagnosis systems for lesion detection can increase diagnosis accuracy, which inherently reduces the number of biopsies made by dermatologists to determine the nature of the lesion and can also aide with early detection of skin cancer like melanoma which can turn fatal if left undiagnosed and untreated.

While designing a diagnosis system for skin cancer that may be distributed to physicians and patients which is by nature image-based, the requirement that the system should be able to run on different types of personal devices like mobile phones and tablets to maximize its access to patients around the world, regardless of hardware or connection capabilities, was established very early. As a result, the diagnosis system offloads the diagnosis process to a cloud-based architecture which drastically reduces the processing burden on client devices.

1.2 PROBLEM STATEMENT

On the research end of deep learning-based diagnosis systems, there is progress being made, but a full, production-level diagnosis system is still lacking in the field of dermatology. As such, the focus of this thesis is the design and implementation of a skin lesion diagnosis system powered by convolutional neural networks and based on cloud architecture. Front-ends for this system can be designed to run on multitude of devices including mobile phones, tablets, and personal computers. A mobile application has also been developed as a front-end to showcase the system.

1.3 MAIN CONTRIBUTION

For this thesis the following have been completed:

1. Deep learning-based classifier that processes user submitted lesion images which runs on a server connected to the cloud database.
2. Deep learning-based classifier that perform quality checks and filters user requests before the request is sent off to the diagnosis classifier.
3. A mobile application that runs on Android and iOS platforms to showcase the system.
4. Cloud database and storage for storing user request and diagnosis results.
5. The design and implementation of the system's overall architecture.

1.4 OVERVIEW OF THESIS

In chapter 2, a background on skin cancer, the magnitude of the problem, and current deep learning-based diagnosis solutions are explored. In chapter 3, the design and implementation of the diagnosis system and its components are explained. In chapters 4

and 5, the architecture, training, and results of the deep-learning-based diagnosis and preliminary classifiers are discussed. The design and implementation of the mobile application demo is examined in chapter 6, and in chapter 7, the thesis is concluded along with a list of future work.

CHAPTER 2

BACKGROUND

In this chapter, the background on skin cancer in United States, the poor accuracy of traditional diagnosis methods, and the reasons why an effective diagnosis system is needed are explored. Next, the state of the art of the current deep learning methods for skin lesion diagnosis are discussed.

2.1 SKIN CANCER IN U.S

Skin cancer is a major medical problem. 5.4 million skin cancer cases occur every year in United States alone [1]. Every year more people are diagnosed with some form of skin cancer than all other types of cancers put together: so much so that one in five people living in United States will be diagnosed with skin cancer in their lifetime [2, 3]. $8.1 billion is spent annually in United States for skin cancer treatment, $3.3 billion of which is the total cost of treatment for melanoma alone [4].

Melanoma is responsible for 75% of annual deaths due to skin cancer, claiming 10,000 lives annually although it makes up only 5% of skin cancer cases [1, 5]. Early detection for melanoma is vital for survival. If detected in its earliest stages, survival rate for melanoma is extremely high at 99%, but falls to 14% in its latest stage.

As shown in Figure 2-1, visual differences between different skin lesions, especially melanoma and benign lesions, can be minute. There are several visual methods used by dermatologist to diagnose melanoma without biopsy, but the accuracy of these

methods is poor, only around 60% [6]. Since these methods are not reliable, biopsies are generally required for diagnosis. As a result, computer aided visual diagnosis systems made available to dermatologists and patients may provide more accurate and convenient methods of diagnosis to improve the rate of early detection of skin cancer and reduce the number of diagnosis tests.

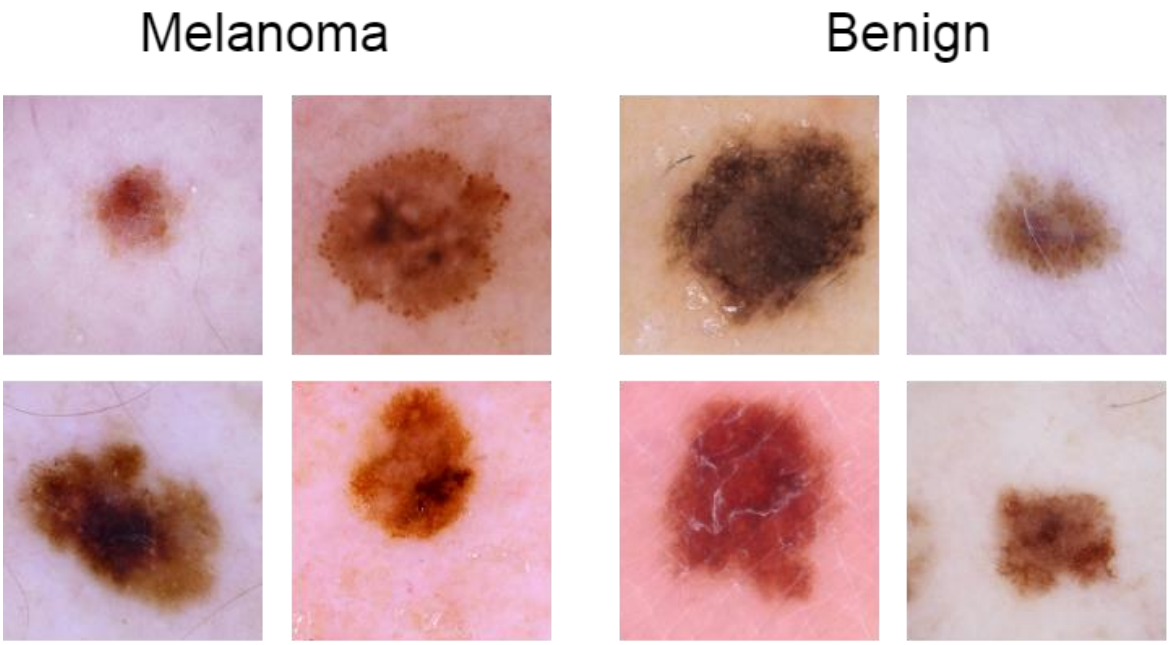

Figure 2-1 Visual similarities between melanoma and benign lesions make it hard for dermatologists and machine learning algorithms to visually identify the lesions. Sample images taken from the "ISIC 2018: Skin Lesion Analysis Towards Melanoma Detection" grand challenge datasets [14,15].

2.2 DEEP LEARNING IN LESION DETECTION

In recent times, deep learning, specifically convolutional neural networks (CNNs), have rapidly become the technique of choice in computer vision. This is due to the contribution of Krizhevsky et al. (2012) winning the ImageNet competition in 2012

and other more recent successes of CNNs improving the state of the art in many domains [7]. To date, CNNs are the most successful type of models for image analysis as highlighted by Litjens et al [8]. This is becoming increasingly the case for many visual problems in the field of medicine as well.

To tackle the problem of skin lesion diagnosis, several authors have used CNNs. Training the CNNs were done using only dermoscopic and clinical images as input along with image labels in a single CNN [9, 10]. In the case of Esteva et al., the trained model was tested against 21 board-certified dermatologists and achieved expert level performance; a feat that, according to the authors, have not been achieved before [10].

One of the general downsides of training neural networks is the requirement for a large training set. Another challenge for this specific task is that images of lesions may vary in factors like lighting and zoom or even feature the same skin disease in different stages which may appear different. Esteva et al. were able to overcome these challenges by collecting almost 130,00 clinical images and 3370 dermoscopy images. They utilized GoogleNet Inception v3 CNN architecture that was pre-trained on 2014 ImageNet Large Scale Visual Recognition Challenge and trained the network with the collected lesion images using transfer learning [11,12,13].

Esteva et al. also point out that with billions of smartphones in use around the world that can be equipped with deep neural networks, low-cost access to diagnostic services can be provided universally. This thesis draws upon that idea and explores an implementation of such a system.

CHAPTER 3

DIAGNOSIS SYSTEM ARCHITECTURE

In this chapter, an overview of the diagnosis system is explained. Detailed explanation of each component is explained in the following sections. As final remark, alternative architectures, along with their advantages and disadvantages, are compared with the implemented architecture.

3.1 OVERVIEW

The diagnosis system allows users to upload lesion images from their personal devices and receive diagnosis results where the diagnosis process is offloaded to server that is running a diagnosis CNN. The user can utilize, if available, the native camera on the device, or upload preexisting images. Any device that has internet connection could connect to the cloud database and storage and upload lesion images to be processed. This includes mobile phones, tablets, and personal computers. The system consists of the client device, cloud database/storage, and server that is connected to both cloud database and the script that loads the CNN as shown in Figure 3-1.

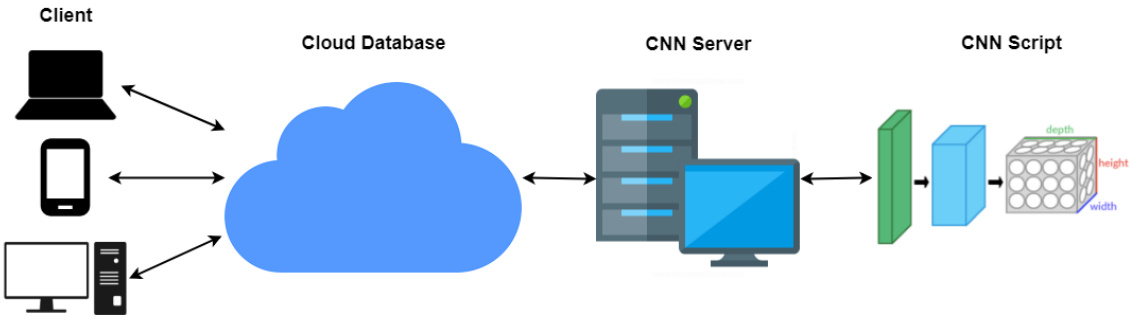

Figure 3-1 The arrows show the flow of the system where user requests are channeled from the cloud database/storage to the CNN server and script, from which diagnosis results flow back to the cloud databases and eventually the client device.

Once an image is uploaded from a device to the cloud, the CNN server is notified which then downloads the image and informs the CNN script to process the image with the already trained diagnosis CNN. In the case of bad user requests, such as images that do not contain lesions or poor-quality images, a preliminary CNN filters requests and controls the flow of the CNN script. Bad requests get reported as such to the CNN server, and good requests are allowed further processing by the diagnosis CNN.

Once the diagnosis CNN processes the image and acquires a skin lesion classification, such as melanoma or benign lesion, and confidence score, the results are reported back to the CNN server which updates the cloud database. Changes made to the cloud database are automatically sent to the client.

3.2 CLIENT

Applications running on client devices may be relatively simple as the only requirements are connection to the cloud database and client-side image resizing. The architecture of the system was designed to keep client memory and processor requirements minimal and

be accessible to a large range of devices with varying specifications such as age, processing power, and memory capacity.

The issue of slow internet upload and download speed is also mitigated by utilizing client-side image resizing which resizes lesion images to the exact size as required by the CNN. As a result, patients and physicians who may have access to different types of devices and internet connection can employ the system with ease.

The application running on devices with cameras can also utilize the camera and handle the task of taking images of lesions. This has been implemented in the mobile phone application written for this thesis. The details of the mobile app are discussed in the Chapter 6.

3.3 CLOUD DATABASE

The task of the cloud database is to establish connection between client devices and the CNN server and store and serve data coming in from both directions. One of its requirements is that it must be able to connect to a large range of devices, store lesion images uploaded from clients into a filesystem and capture results for these images sent from the CNN server into a database. These stored results must also be streamed back to the client device as soon as they become available.

Google's Firebase has many services that offers these features, so we elected to build our system with it, specifically with Cloud Firestore and Cloud Storage Firebase services [16]. Firebase provides client and server-side libraries for applications that run on Android, iOS, and JavaScript, allowing mobile, web, and server applications to integrate with this system. Cloud Storage handles the storage and serving of files which

are lesions images in this case. Firestore is a real-time, stream-based NoSQL database service that synchronizes data events across applications.

Data synchronization is vital to this system because as users upload requests to Firestore, the CNN server is synchronized with the upload requests, and immediately initiates the diagnosis process. In the other direction of data flow, once Firestore is updated with diagnosis results from the CNN server, the client device is then synchronized with the new results as soon as they are available.

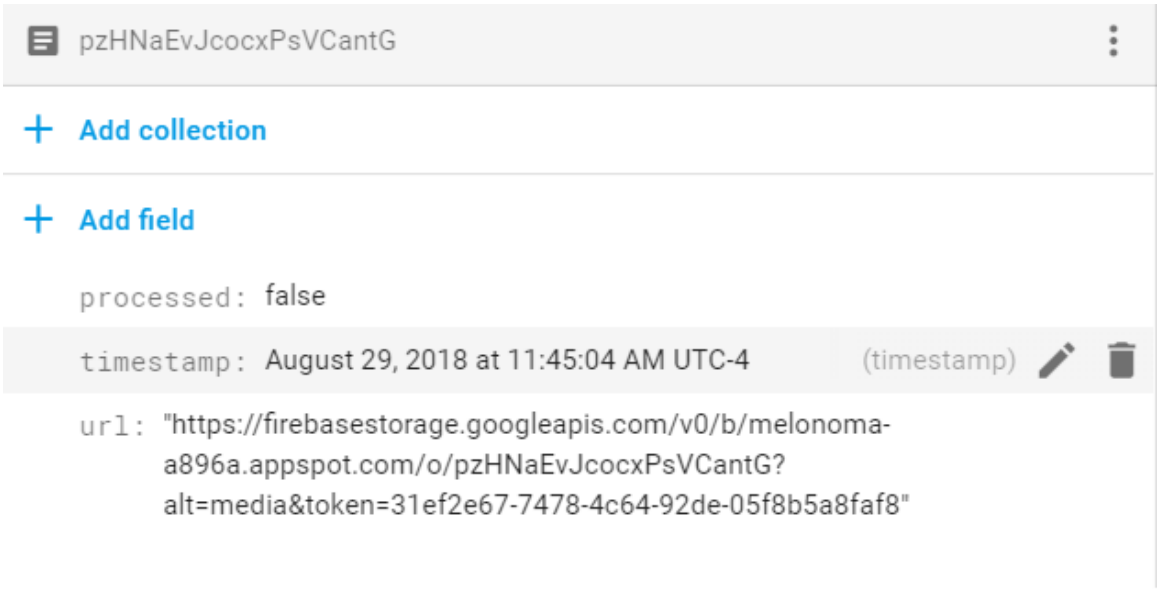

Figure 3-2 A screenshot of a Firestore document that has not yet been processed by the CNN server and CNN script.

When a diagnosis for an image is requested, a new document with a unique id is created in Firestore, and the image is uploaded with this id as the filename to cloud storage. The document initially only stores the id, timestamp, and the file URL of the

uploaded image in Cloud Storage as shown in Figure 3-2. In the next step, the CNN server gets notified of the document and handles the rest of the process.

3.4 CNN SERVER

The CNN server is written in JavaScript for Node.js, and acts as a task manager for the CNN script. Since Firestore synchronizes all connections, the CNN server which is already listening for changes in Firestore is notified immediately of requests in the form of newly created documents. The server then retrieves documents that have not yet been processed, and using the id of the documents, downloads the corresponding lesion images as shown Section 2 in the source code of the Node.js server below:

```
const admin = require('firebase-admin');
const keys = require('./key.json');
const spawn = require('child_process').spawn;
const fs = require('fs');

admin.initializeApp({
    credential: admin.credential.cert(keys),
    databaseURL: "https://melonoma-a896a.firebaseio.com",
    storageBucket: "gs://melonoma-a896a.appspot.com"
});

const db = admin.firestore();
function processUpload(id) {
    jobs[id] = true;
    py.stdin.write(JSON.stringify({id}) + '\n'); //send off to script
to process
}

let jobs = {};

// 1) spawn the CNN script
```

```
const py = spawn("python", ["-u", "app.py"]);
py.stdout.setEncoding('utf-8');
py.on('exit', () => {
    console.log('PYTHON SCRIPT ERROR');
});

// 2) listen for new documents in Firestore and download images
db.collection('uploads').where('processed', '==', false)
  .onSnapshot(docs => {
    docs.forEach(doc => {
        if (jobs[doc.id] !== undefined) return;
           admin.storage().bucket().file(doc.id).createReadStream()
           .pipe(fs.createWriteStream(`tmp/${doc.id}`).on('finish', ()
=> {
             processUpload(doc.id);
        }));
    });
});

// 3) returned results from the CNN script
py.stdout.on('data', (data) => {
    let result = JSON.parse(data); // json result
    result.processed = true;
    console.log(result);
    fs.unlink(`tmp/${result.id}`, (error) => { /* handle error */ });
// delete file
    db.doc(`uploads/${result.id}`).update(result).then(() => { //
update change
        delete jobs[result.id];
    });
});
```

```
process.stdin.resume();
```

Using the document ids, the uploaded images stored in Cloud Storage are downloaded into a temporary location in the filesystem. The CNN script has access to the temporary file location so that images can be loaded and preprocessed before running them through the CNNs.

The CNN script is a child process of the CNN server and is spawned at the initialization of the server in Section 1 of the source code. The communication between the script and server is done using standard IO between parent and child processes. Notifying the CNN script to processes a request, handled by the processUpload function, is done by writing the id of the document to the standard input of the CNN script.

In Section 3, results from the script in the form of stringified JSON are written to the standard input of the server. Next, the JSON response is parsed, and with the extracted id, the corresponding document in Firestore is updated with the results from the script. For clean-up, the downloaded image in the temporary location is deleted. As shown in Figure 3-3, the updated document includes the label of the lesion, confidence score of the CNN for the diagnosis, and a boolean for recording the malignancy of the lesion. After this update, the client device is synchronized with the changes in Firestore and receives the results of the diagnosis.

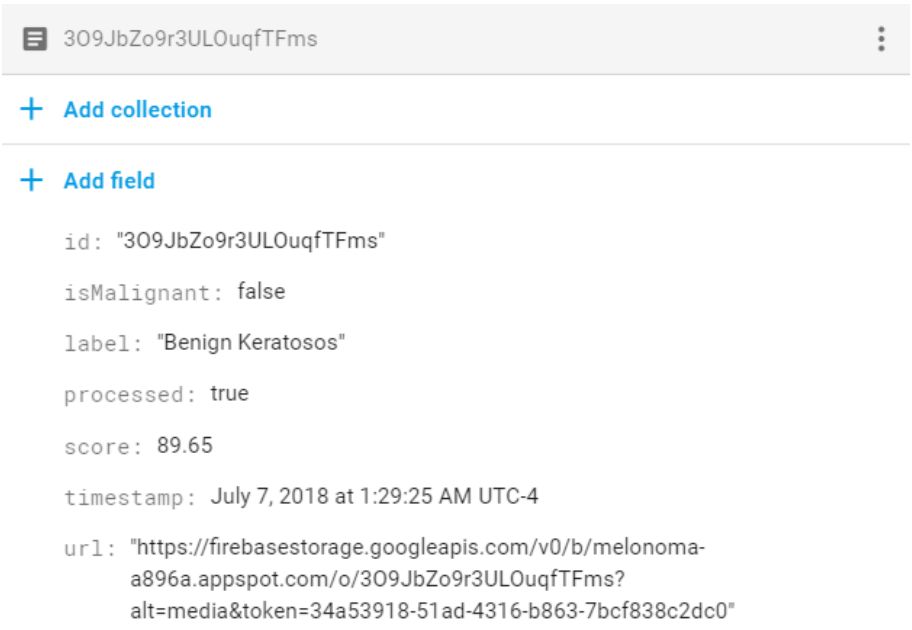

Figure 3-3 A Firestore document with the updated by the CNN server with results from the CNN script.

The CNN script is a child process spawned by the CNN server that loads the trained CNN model to process the uploaded lesion images and returns the output from the CNN. The script also features a preliminary CNN that was trained to detect images that do not contain lesions or meet a certain quality level. Images that do not pass the preliminary CNN are rejected, and do not get processed by the diagnosis CNN. In that case, the CNN script returns an error message to the CNN server which then updates the Firestore document in the Cloud Database with the appropriate fields as can be seen in Figure 3-4.

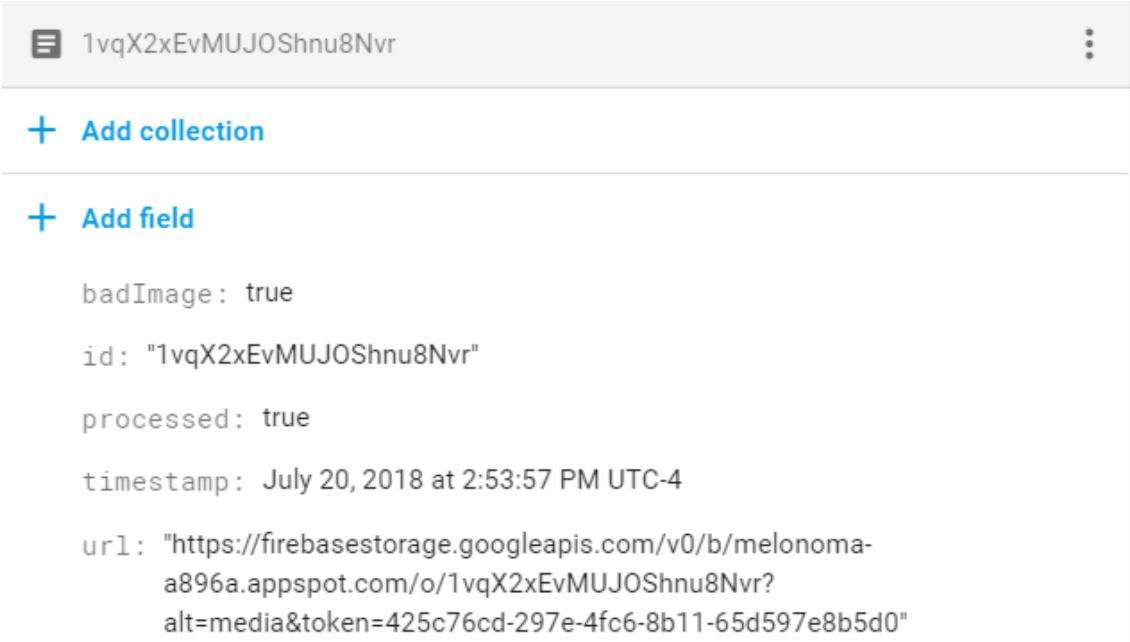

Figure 3-4 A Firestore document that reports that an image request was declined by the preliminary CNN.

The preliminary CNN is tiny in number of parameters compared to the diagnosis CNN, and a quick check by this CNN increases the overall quality of the system and may even increase the overall efficiency of the system by preventing some images making their way to the much larger, slower diagnosis CNN. The flow of the script can be observed in Figure 3-5.

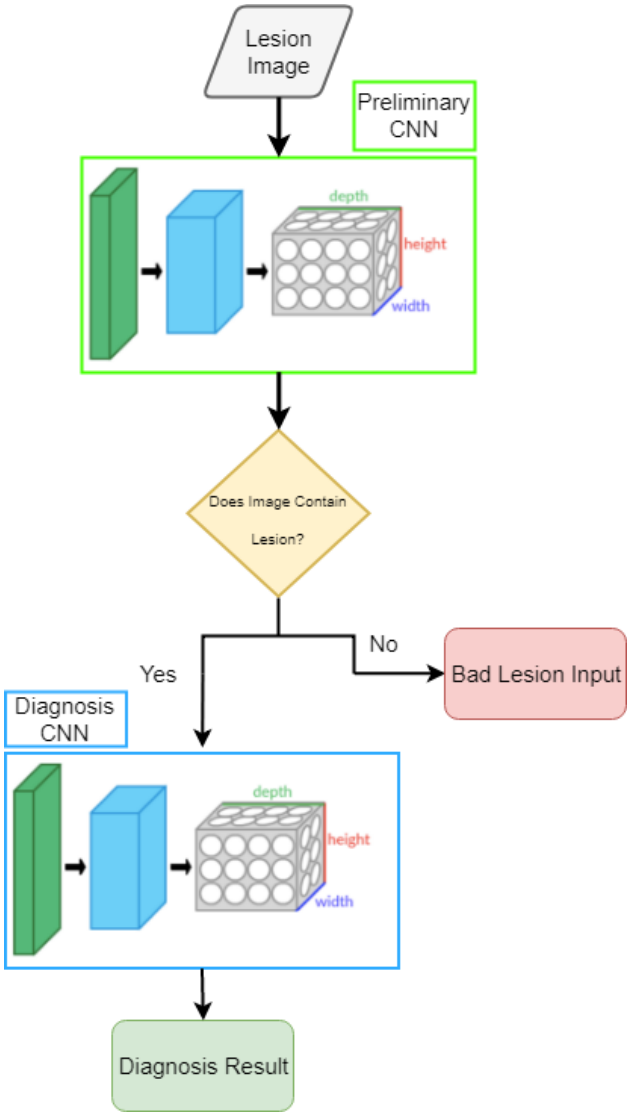

Figure 3-5 The preliminary CNN prevents images that do not contain lesions from getting processed by the diagnosis model.

The script was written in Python, and for training and running the CNNs, the neural network library Keras was used [17]. The source code for the CNN script in its entirety can be observed below:

```
import sys
import json

from keras.applications.resnet50 import preprocess_input
from keras.models import load_model
from keras.preprocessing import image
import numpy as np

# 1) Two trained models
model = load_model('models/model_resnet_best.hdf5')
isLesionModel = load_model('models/model_islesion.hdf5')

# 2) Labels CNN outputs
LABEL_STRS = ["AKIEC", "BCC", "BKL", "DF", "MEL", "NV", "VASC"]
LABEL_FULL = ["Actinic Keratosis", "Basal Cell Carcinoma",  "Benign
Keratosos", "Dermatofibroma", "Melanoma", "Nevus", "Vascular Lesion"]
ISMALIGNANT = [True, True, False, False, True, False, False]

# 3) Await IDs input from CNN server
for line in sys.stdin:

    # 3.1) Extract id from JSON string from server and load image
with id
    id = json.loads(line)['id']
    PATH = "tmp/" + id
    img =
np.array([preprocess_input(image.img_to_array(image.load_img(PATH)))]
)

    # 3.2) Run the image through the preliminary CNN to check if
image contains lesion.
    pred = isLesionModel.predict([img])
```

```
    score = round(max(pred[0]) * 100.0,2) #score
    idx = pred.argmax(axis=-1)[0] #label index

    # 3.3) if image is good, run the diagnosis CNN, else report back
with badImage label
    if idx == 0:
        result = {'badImage': True, 'id': id}
        print(json.dumps(result))
    else:
        pred = model.predict([img])
        score = round(max(pred[0]) * 100.0,2) #score
        idx = pred.argmax(axis=-1)[0] #label index
        result = {'score': score, 'isMalignant': ISMALIGNANT[idx],
'label': LABEL_FULL[idx], 'id': id}
        print(json.dumps(result))
```

In Section 1 of the source code, the two models are loaded. The diagnosis model was trained on the ISIC 2018 dataset which contains images for seven skin diseases including melanoma as can be observed in the array of labels in Section 2 [14,15]. The preliminary CNN model has two outputs for lesion and non-lesion images. The datasets used to the train the network are Caltech 101 for non-lesion and the ISIC dataset for lesion images [18].

To begin the diagnosis, several things need to occur. First, the id of the request is needed. Next, using the id, the image corresponding to the id in the temporary location in the filesystem gets loaded and goes under preprocessing to become ready to be used as input to the primary CNN and eventually diagnosis CNN.

The request ids are passed to the standard input of the script in JSON form from the CNN server, and in Section 3.1, the id is loaded from the JSON input, and the image corresponding to the id is decoded and preprocessed for both CNNs. Next, in Section 3.2, the input image is processed by the preliminary CNN, and an output, either lesion or non-lesion, is acquired. In the following subsection, images labeled as good images by the preliminary CNN are processed by the diagnosis CNN, and the diagnosis results are printed out in JSON form back to the CNN server. Non-lesion images on the hand are reported back with an error field. The discussion on the architecture of the models and the training methods take place in the next chapter.

3.5 ALTERNATIVE ARCHITECTURES

There are several alternative architectures that were considered during the design stage of the system. Generally, there are two ways of incorporating deep neural networks in a mobile app: running the network locally on the native device or offloading the processing to servers. Running locally only became possible in recent years with leaps in processing power in mobile devices as training and running neural networks require huge amounts of computation power. In fact, one of the main reasons for the booming of deep learning in general is due to the recent growth in processing capabilities of processors, mainly graphics processing units (GPU) [19]. In addition to computation demands, the size of the model can put restraints on the device's memory.

ResNet50 is the model that was used for the diagnosis CNN, and its binary size is roughly 99 Megabytes (MBs) with 25,636,712 parameters which makes it difficult to run on older mobile devices with constrained memory and computation capabilities [20, 21]. Another downside of running locally is that some application that are powered by neural

networks may need frequent updates. Any updates to the core model because of further training as more and more medical data is gathered would need be downloaded by the client device in the form 99 MBs patches in the least, putting users with weak and slow Internet connection out of reach.

For server architectures, a seemingly simpler, monolith architecture was also considered where client requests, database, and the CNN are handled by a single server instance. The cloud architecture, on the contrary, establishes connection between specialized programs written in different programming environments that solve different tasks and does not force the entire system to operate in one environment, as it is the case in a monolith server. It follows a software philosophy where a large system is composed of several components or programs that performs a simple task well and pass on data to another component to complete another task.

In a monolith set up, the flexibility of using specialized tools and programming environments that perform a task well outside of the monolith environment is constrained. Generally, a single programming language or environment are preferred in a monolith architecture, and as a result, although the monolith architecture has fewer independent components than the cloud architecture, the implementation may be more complex and inefficient.

An example of specialization is that many neural network libraries have been written in Python which makes Python a popular, specialized language of choice for implementing and training neural networks. Node.js is also another popular programming environment used for creating servers which is featured in this cloud architecture. The

cloud architecture takes advantage of segregating tasks to specialized components whereas a monolith system offers very little flexibility.

The cloud architecture can also grow more naturally and efficiently. To respond to high user demand, multiple CNN server nodes with same cloud database can be spawned in different geographical locations where each CNN server is also capable of spawning multiple CNN scripts to maximize throughput with little change to the source code.

Data access is also handled better in a cloud set up. Data storage and processing are separated in the form of the cloud database and CNN server, so users attempting to access results from the cloud database are not affected by any amount of request traffic the CNN server may be under. Because of the overwhelming benefits of the cloud architecture, this architecture was opted to be implemented.

CHAPTER 4

DIAGNOSIS CNN

In this chapter, the dataset that was used to train the diagnosis CNN and the rebalancing of the dataset to stabilize network performance is discussed. Next, the architecture of the CNN and training paradigms used in experiments are explored. The chapter is concluded with the accuracy results of the network, and a brief discussion on comparison of the network with human performance.

4.1 DATASET

The dataset containing lesion image samples used to train the preliminary and diagnosis CNNs come from the "ISIC 2018: Skin Lesion Analysis Towards Melanoma Detection" grand challenge datasets [14,15] which contains a total of 10,015 samples of seven skin disease types shown in Table 4-1. The dataset is not balanced as there are significantly more nevus (benign, non-harmful) samples than any other samples combined. Also, dermatofibroma and vascular lesion are severely underrepresented. An unbalanced dataset can make it difficult to train and assess the performance of a network, so the dataset was rebalanced before training.

Label	Count	%
Actinic Keratosis	327	3.3
Basal Cell Carcinoma	514	5.1
Benign Keratosis	1099	11.1
Dermatofibroma	115	1.1
Melanoma	1113	11.0
Nevus	6705	67.0
Vascular Lesion	142	1.4

Table 4-1 Skin diseases and sample counts in the ISIC 2018 dataset (10,015 samples).

There are in fact multiple ways of assessing the performance of a network. For some classification tasks, accuracy may be a sufficient metric. It can be used to evaluate the general performance of the model. Following Table 4-2, accuracy can be calculated using equation (4.1). It is essentially the ratio of correctly identified positive and negative labels to the total number of samples in the dataset. Accuracy largely ignores the balance between false negatives and false positives percentages. Two networks with similar True Positive (TP) and True Negative (TN) rates but different False Positive (FP) and False Negative (FN) rates will generally report similar accuracy. If FN and FP rates are significant metrics for a task, then the accuracy metric won't give enough information about the performance of the model.

$$\frac{TP + TN}{TP + TN + FP + FN}$$

Equation 4.1 Accuracy.

	Truth - Positive	*Truth - Negative*
Prediction - Positive	True Positive	False Positive
Prediction - Negative	False Negative	True Negative

Table 4-2 A visualization of 2x2 confusion matrix.

A low FN percentage, which indicates that only a small of patients who have the disease are wrongly diagnosed as being clear of it, is important for medical diagnosis tasks, especially in fields where deadly diseases are dealt. A low FN inherently results in a high FP rate which indicates that a higher number of people who do not have the disease are wrongly diagnosed as having it. Having a high FP, but low FN rate is a vital and necessary tradeoff for most medical diagnosis systems.

Sensitivity is a performance metric in addition to accuracy that is inversely proportional to FN and can be calculated using equation (4.2). Diagnosis system generally are designed to have high sensitivity values. To increase the sensitivity value, FNs must be decreased.

$$\frac{TP}{TP + FN}$$

Equation 4.2 Sensitivity.

Tweaking a model's sensitivity rate can be achieved with rebalancing the dataset so that during training the network can bias certain overrepresented labels. The original dataset contains far more nevus samples than any other category, but sensitivity rating of

melanoma is far more important than consistently classifying benign lesion. As a result, rebalancing is necessary to easily assess and tweak a model's performance for melanoma.

Before training. the dataset was split to create train and test sets. 80% of the dataset was used for training and 20% for testing. Each split set was rebalanced by under-sampling nevus samples so that only 20% of each set, train and test, contained nevus samples. After balancing, melanoma became one of the most represented diseases in the training and test dataset which helped increase the sensitivity of the network for melanoma. The balance of the training and training sets can be observed in Tables 4-3 and 4-4.

Label	Count	%
Actinic Keratosis	265	8
Basal Cell Carcinoma	408	12.3
Benign Keratosis	893	27.0
Dermatofibroma	93	2.8
Melanoma	874	26.4
Nevus	662	20.0
Vascular Lesion	112	3.4

Table 4-3 The rebalanced train set.

Label	Count	%
Actinic Keratosis	62	7.5
Basal Cell Carcinoma	106	12.8
Benign	206	24.8

Keratosis		
Dermatofibroma	22	2.7
Melanoma	239	28.8
Nevus	166	20.0
Vascular Lesion	30	3.6

Table 4-4 The rebalanced test set.

4.2 CNN ARCHITECTURE AND TRAINING

The dataset is relatively small which makes it hard to train a neural network from scratch, and even made harder by the fact that the dataset was rebalanced by under-sampling overrepresented categories and split to create training and testing sets. Because training from scratch isn't plausible, transfer learning on the ResNet50 architecture pre-trained for the 2014 ImageNet Large Scale Visual Recognition Challenge was used for training [13, 21].

Transfer learning refers to customizing an already trained network on a different dataset for a different task to solve the another classification task. CNNs contain several convolutional layers, generally some fully connected layers, if any at all, and a final softmax layer used for classification. The task of the convolutional layers is generally to build up a deep feature extraction of the image input. Early layers might extract primitive features like lines or edges in the image where later layers will map these features to form more complex features like object shapes. The final fully connected layers perform a final mapping, usually to prepare the features for classification, before the softmax layer.

A transfer learning operation on a CNN usually involves replacing the final fully connected and softmax layers with a new set of fully connected layers and a softmax

layer. To train the network, input is forward propagated as normal, but during backpropagation, only the newly added final layers' weights are updated. In other words, the pretrained network acts as a feature extractor for the new layers where the new layers use these features for classification.

ResNet became popular when it won the ImageNet Challenge in 2014. The architecture appears to be a simple, sequential CNN with 3x3 convolutional filters with 1x1 and 2x2 strides, but it owes its success to skip connections that it popularized. The output of every second convolutional block has an extra, direct connection to the input of the next, second block as can be seen in Figure 4-1. These skip connection channel features from previous convolutional layers directly into later layers which helps reduce the saturation of the accuracy of the network as the number of layers grow. The ResNet architecture allows the design of truly deep networks without loss of accuracy.

The final convolutional layer a ResNet network connects to a flattened average-pooled 2048-dimension layer which is connected to the final, softmax dense layer of the network which output probabilities for different categories. The final softmax layer, per transfer learning, was replaced by two densely connected layers with 500 neurons in each and a new, seven category softmax layer. The convolutional layers in ResNet act as features extractors for these new layers.

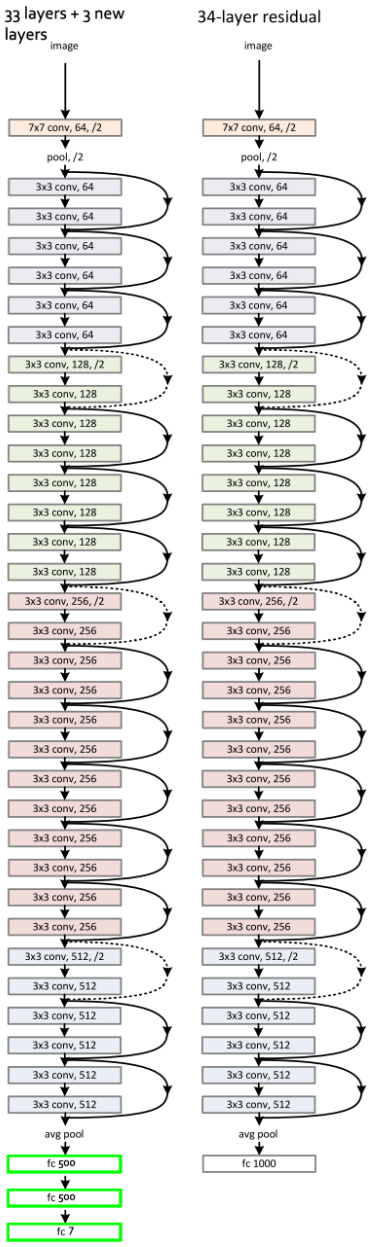

Figure 4-1 The last layer of the network originally designed for ImageNet (right network) is replaced with three new layers (left network, green blocks). The networks in the figure contain 33 convolutional blocks, but the diagnosis CNN contains 49 [21].

During training, the entire network besides the newly added layers were kept frozen. For regularization, 25% dropout between every connection and batch

normalization before every activation, to speed up network optimization, was added to the new layers [22,23].

Dropout is a regularization technique for combatting overfitting during training to improve generalization of the network. In dropout, given probability p, neurons are randomly ‘dropped’ from selected layers and and have no output to be used in forward or backward propagation. For every batch of training data, dropout is reapplyed randomly. With each batch, different sets of neurons are trained together which regularizes the network and reduces overfitting.

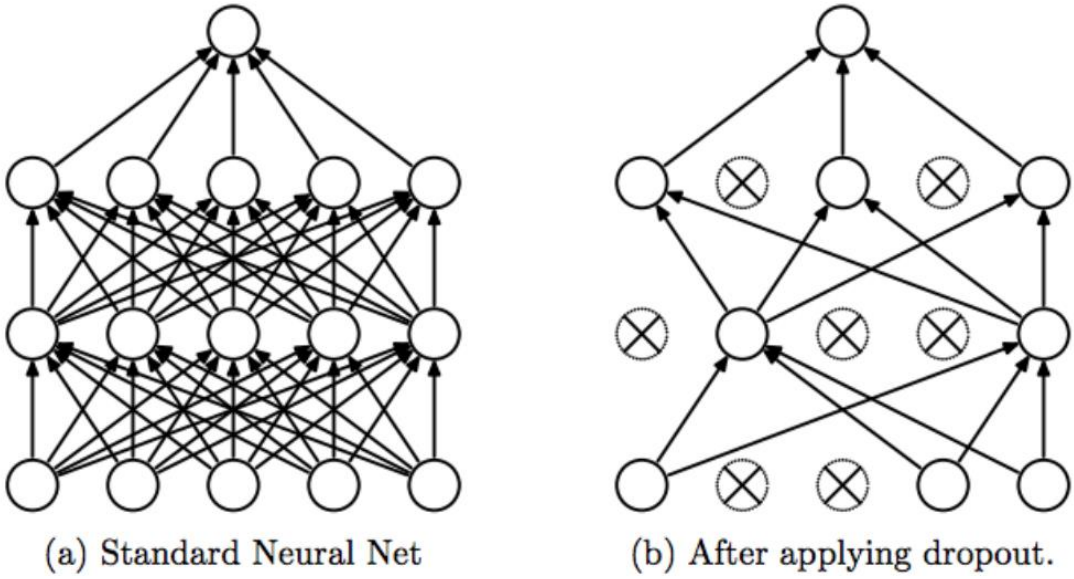

Figure 4-2 During training, dropout is used to randomly select neurons to be used in forward and backward propagation

Batch normalization can be fitted after each layer so that the output of the layer has zero mean or unit variance. Using the algorithm defined in Figure 4-3, the output of a layer for a batch is normalized before the next layer uses this output as input. Batch normalization comes with two trainable parameters in the final scale and shift step which

get updated during backpropagation. Using normalization to limit the distribution of the activations in the hidden layers helps the network train faster and reach optimum level more quickly.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 4-3 By normalizing the range of layer outputs with batch normalization, the network trains more quickly.

Lastly, stochastic gradient descent (SGD) with a stable learning rate of 0.01 was used to train the CNN. The following code sample contains the implementation and networks parameters of the new dense layers writing with Keras:

```
epochs = 350
drop = 0.25
dns = 500
```

```
batch_size = 16
rg = 1e-3

mc_top = ModelCheckpoint('weights/model_bottleneck_test.hdf5',
monitor='val_acc', verbose=0, save_best_only=True,
save_weights_only=False, mode='auto', period=1)

model = Sequential()

model.add(Flatten(input_shape=train_data.shape[1:]))
model.add(Dropout(drop))

model.add(Dense(dns, kernel_initializer='he_normal',
kernel_regularizer=l2(rg)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(drop))

model.add(Dense(dns, kernel_initializer='he_normal',
kernel_regularizer=l2(rg)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(drop))

model.add(Dense(num_classes, kernel_initializer="he_normal",
activation='softmax'))

opt = SGD(lr=0.01)
model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_data, train_labels,
          epochs=epochs,
```

```
            verbose=2,
            batch_size=batch_size,
            shuffle=True,
            validation_data=(test_data, test_labels),
            callbacks=[mc_top])
```

4.3 CNN RESULTS

The accuracy of the network on the test set after training is 77.4%. The rebalance of the dataset resulted in high accuracies for non-nevus categories, as shown in Figures 4-4 and 4-5, and having more samples of melanoma than nevus also helped improved sensitivity of the network (lowered false negatives), which is especially vital for diagnosis systems, when diagnosing melanoma vs nevus lesions.

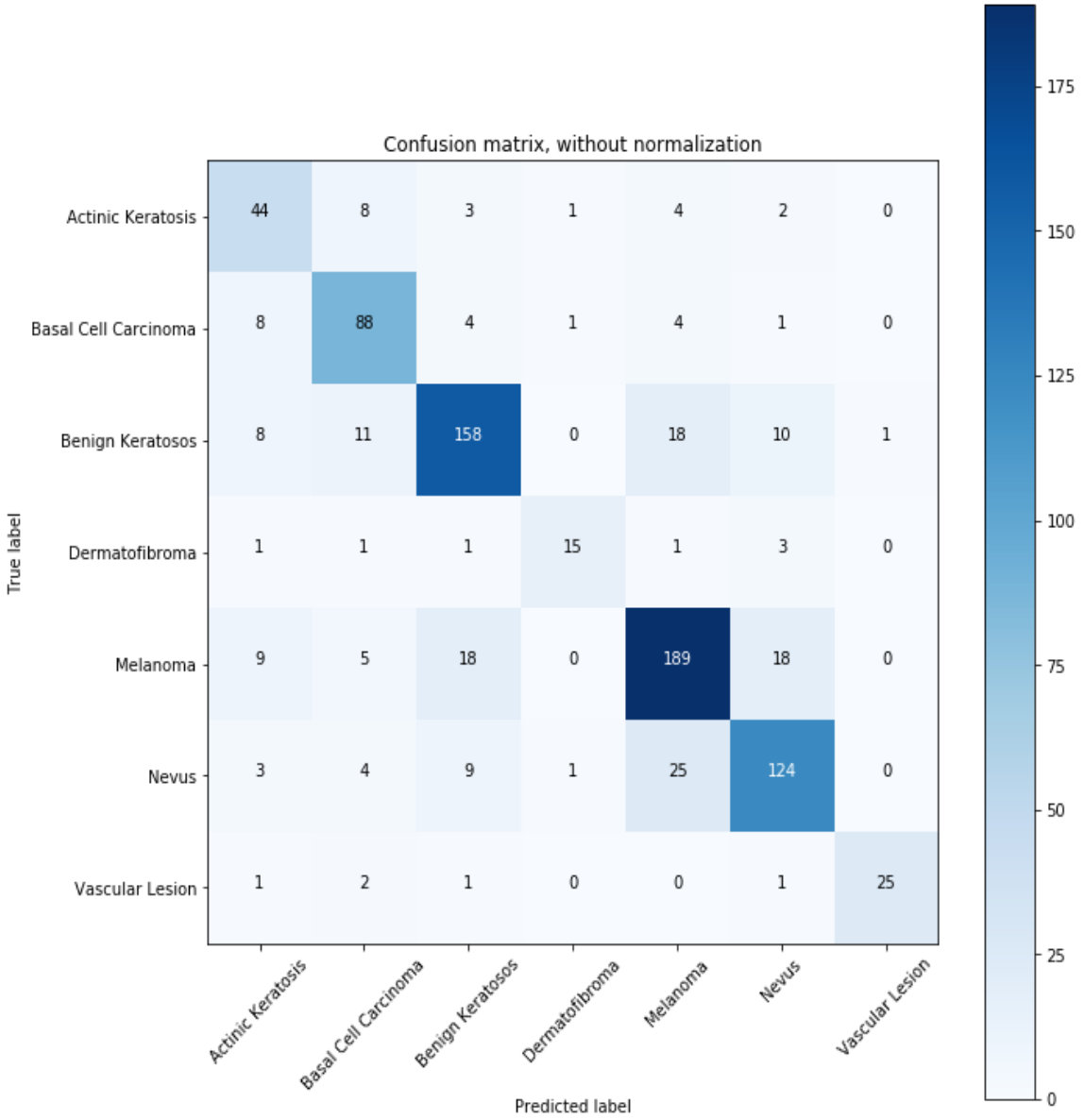

Figure 4-4 Confusion matrix of the test set results

Although human level accuracy on this dataset has not been determined, it is safe to assume that per [4], the accuracy of the diagnosis network, at least on melanoma detection, is on or very close to human level performance.

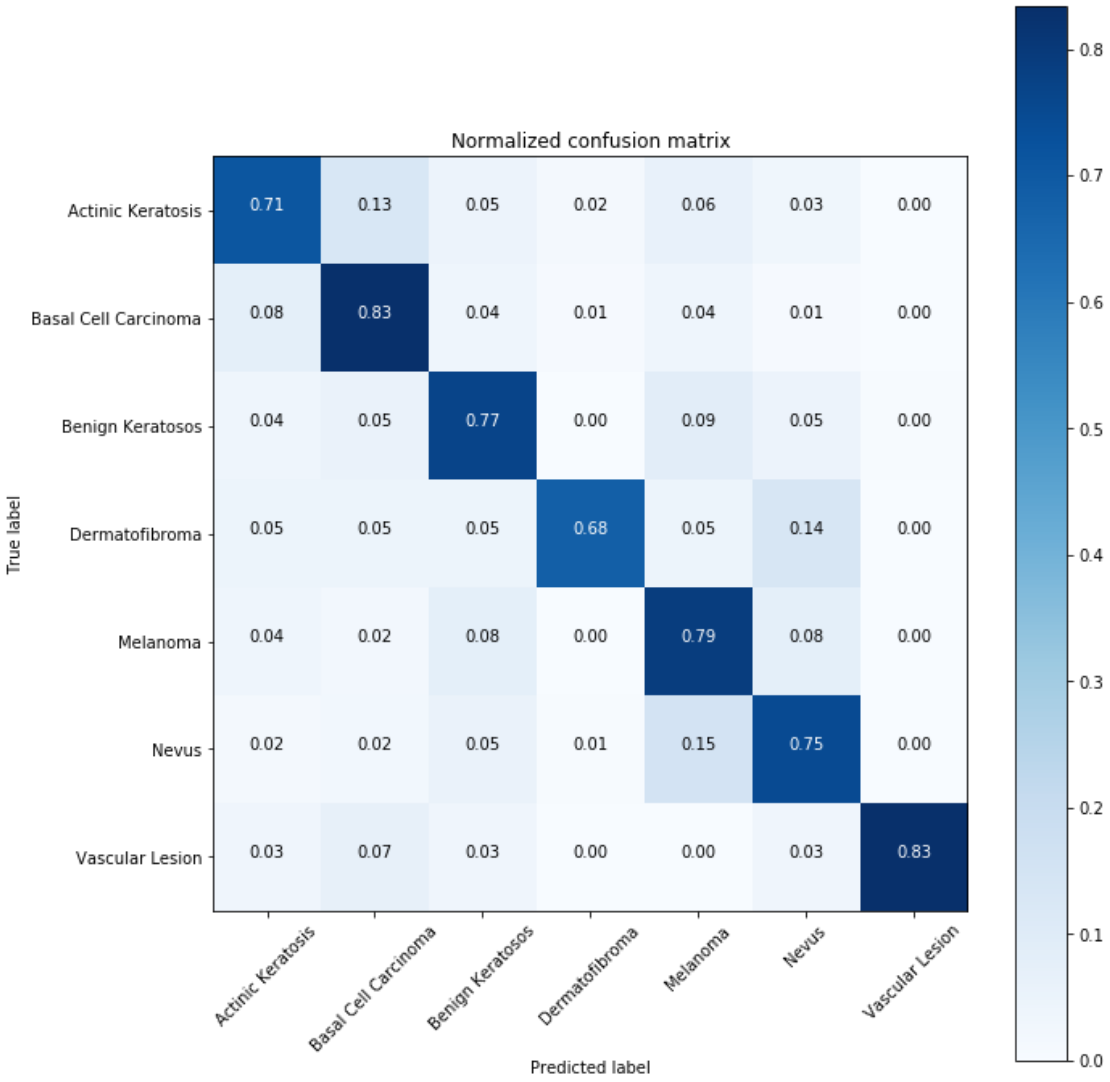

Figure 4-5 Normalized confusion matrix of the test set results (sensitivity).

Further experiments like unfreezing final few convolutional layers to continue backpropagation over ResNet resulted in overfitting and lowered test accuracy. Despite a tiny amount of training data, several CNNs ranging from few layers to up to 18 layers were experimented with, and the best accuracy achieved was 66%. In the end, the network trained with transfer learning was chosen as the diagnosis CNN.

CHAPTER 5

PRELIMINARY CNN

The purpose of the preliminary CNN is to filter bad requests from the user. This way, the diagnosis system won't attempt to classify images that are not lesion images or do not meet the quality of the images in the training set and send back garbage results as a result. To train the CNN, datasets containing categories of common office or household objects that a user might take a picture of to test the system, or any category that is exclusively not medical and skin lesion related to train against the lesion images could work.

As counter to the ISIC 2018 dataset, the Caltech 101 dataset along with ISIC 2018 was used to train the CNN. Essentially, images from Caltech 101 dataset were labeled as 'non-lesion' images and images from ISIC 2018 dataset were labeled as 'lesion' images. The preliminary CNN therefore outputs two probabilities. Samples in the training and testing sets were randomly chosen and contain roughly 8000 and 2000 sample images, respectively.

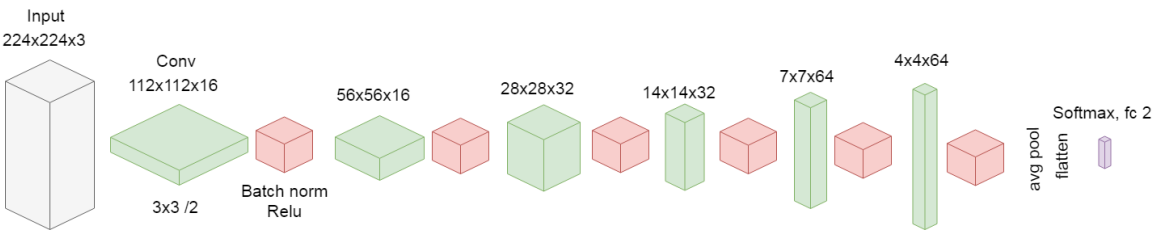

Figure 5-1 The overall architecture of the preliminary CNN.

The architecture of the CNN is relatively small, containing roughly 73,000 parameters. After the input layer, there are 6 convolutional layers with kernel size of 3x3 and strides of 2x2. After the final convolution layer, an average pool is computed, and the layer is flattened which then gets connected to the final double-output softmax layer.

Training was done with SGD and learning rate of 0.01. The architecture can be observed in detail in the Keras code sample below and in Figure 5-1:

```
ROW_AXIS = 1
COL_AXIS = 2
CHANNEL_AXIS = 3

def bn_rel (conv):
    norm = BatchNormalization(axis=CHANNEL_AXIS)(conv)
    return Activation("relu")(norm)

def conv2(prev, filters=64, kernel_size=(3,3), strides=(2,2),
padding='same'):
        return Conv2D(filters=filters,
                      kernel_size=kernel_size,
                      strides=strides,
                      padding=padding,
                      kernel_initializer='he_normal',
                      kernel_regularizer=l2(1.e-4))(prev)
dim = 224
num_classes = 7

# input block
input = Input(shape=(dim,dim, 3))

filters = 16

conv = conv2(input,filters=filters)
actv = bn_rel(conv)
conv = conv2(actv, filters=filters)
actv = bn_rel(conv)

filters = 32
```

```
conv = conv2(actv, filters=filters)
actv = bn_rel(conv)
conv = conv2(actv, filters=filters)
actv = bn_rel(conv)

filters = 64

conv = conv2(actv, filters=filters)
actv = bn_rel(conv)
conv = conv2(actv, filters=filters)
actv = bn_rel(conv)

block_shape = K.int_shape(actv)
pool = AveragePooling2D(pool_size=(block_shape[ROW_AXIS],
block_shape[COL_AXIS]), strides=(1, 1))(actv)
flat = Flatten()(pool)

dnse = Dense(2, kernel_initializer="he_normal", activation="softmax")(flat)

model = Model(inputs=input, outputs=dnse)

opt = SGD(lr=0.01)
model.compile(optimizer=opt, loss='binary_crossentropy',
metrics=['accuracy'])
model.summary()
```

Since images in both categories are very different from each other, the accuracy of the network turned out to be very high. After only three epochs, the accuracy on the test set was 99.4% so training was stopped early, and the preliminary CNN was fitted with this network.

CHAPTER 6

MOBILE PHONE APPLICATION

To showcase the diagnosis system, an application has been developed for both Android and iOS platforms. The application connects to Firebase at startup to upload images and receive results from the diagnosis pipeline. The application is a hybrid mobile app built using Ionic SDK [24]. With Ionic, developers can build mobile and web applications using web technologies like HTML, CSS, and JavaScript along with native features that the device offers like camera and geolocation. Using the same source code, the application can be compiled to run on multiple platforms which includes Android, iOS, and web browsers.

Figure 6-1 The application utilizes the native camera on the mobile phone.

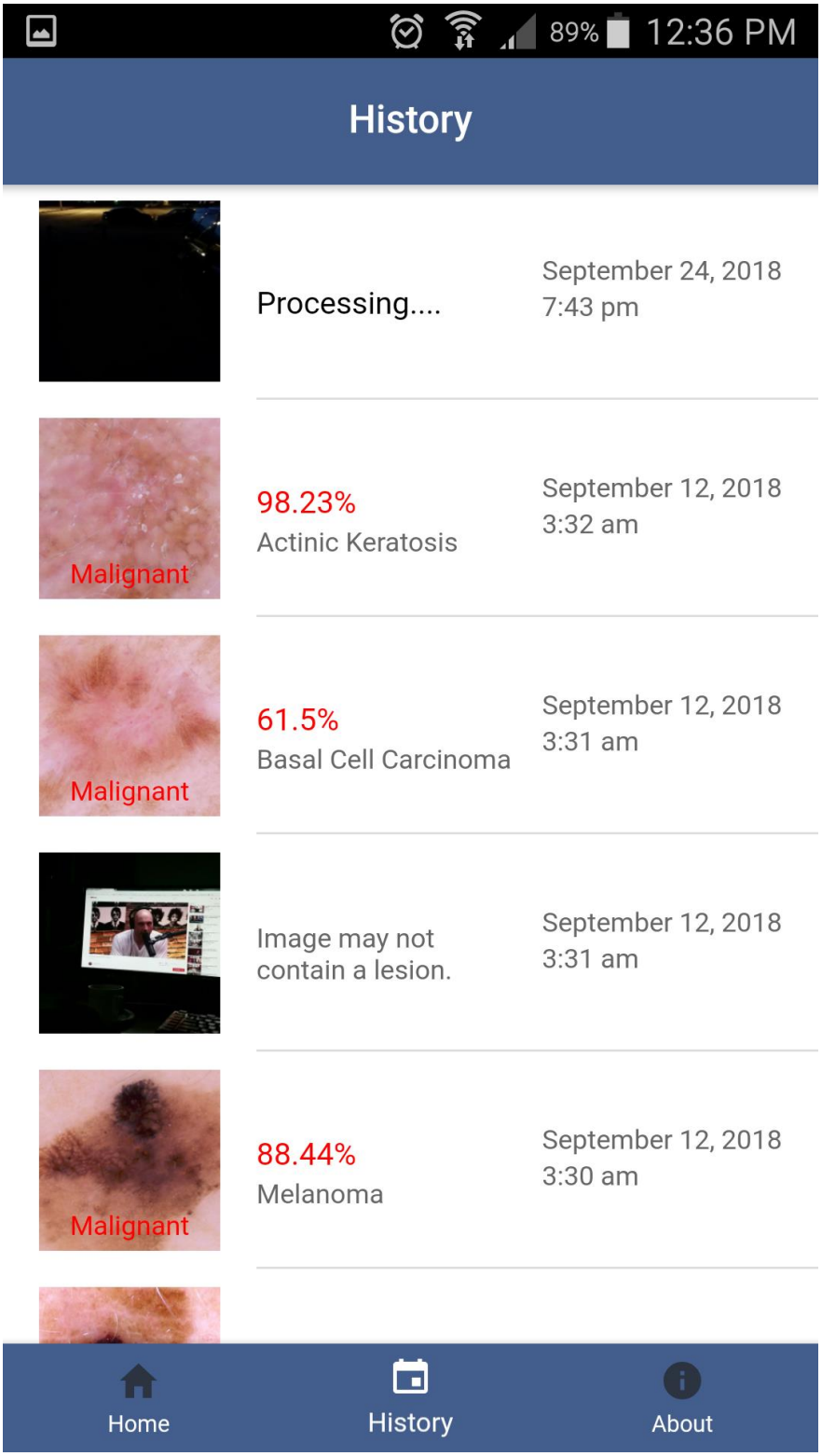

Figure 6-2 History of results page displays diagnosis results.

The application connects to the device's camera and displays the camera stream. The user is guided with a red circle to help place the skin lesion centered horizontally and vertically in the visual field of the camera (Figure 6-1). This is an important feature because the lesion images in the dataset are centered on the lesion and replicating this property in the user requests may increase the accuracy of the diagnosis.

Once an image is taken by the user, the users have the option to either retake the image or upload the image to Firebase. Before uploading, the image is cropped and resized to match the exact dimensions of the diagnosis and preliminary CNNs. Once uploaded, the user is taken to the History page which displays results from the diagnosis CNN.

During the diagnosis process, the uploaded request is marked as 'processing…' as shown in Figure 6-2. Once a result is acquired, the app is updated with a diagnosis and confidence score from the diagnosis CNN. If an image does not pass the preliminary CNN, the user is notified with a warning text.

The diagnosis system being cloud-based, complexity of client application is minimal. The main requirements of the application are connection to the camera, Firebase, and client-side image resizing which can be done with few lines of JavaScript. The minimalist nature of the front-end application also allowed for a very quick design and implementation.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Skin cancer is a serious medical problem. Early diagnosis for type of cancer like melanoma is vital for survival. For this thesis, a skin lesion diagnosis system was built to help with diagnosis. The system is cloud-based and uses Firebase. The diagnosis process is offloaded to servers which allows developers to build front-ends for a large range of mobile devices that are lightweight and minimalistic in code complexity. The diagnosis is handled by a two-stage CNN pipeline where a preliminary CNN performs quality check on user requests, and a diagnosis CNN, whose accuracy is near dermatologist level, outputs probabilities over seven different lesion categories corresponding to the categories in the ISIC 2018 dataset used for training. For training, transfer learning was applied to a ResNet50 network trained on the ImageNet competition dataset.

For future work, to improve accuracy of the diagnosis network, more data would need to be gathered. The diagnosis CNN was trained using transfer learning with roughly 3,000 images and that is not nearly enough to allow experimentation with different CNN architectures and training paradigms. One experiment, put under future work, is combining the diagnosis ResNet with a smaller network that is trained from scratch in multi-stream fashion where the outputs of both networks are connected to a combined final layer. This way, a unique version of ensemble of networks may result in higher accuracy and better generalization.

BIBLIOGRAPHY

[1] Rogers HW, Weinstock MA, Feldman SR, Coldiron BM. Incidence estimate of nonmelanoma skin cancer (keratinocyte carcinomas) in the US population, 2012. JAMA Dermatol 2015; 151(10):1081-1086.

[2] Cancer Facts and Figures 2018. American Cancer Society. https://www.cancer.org/content/dam/cancer-org/research/cancer-facts-and-statistics/annual-cancer-facts-and-figures/2018/cancer-facts-and-figures-2018.pdf. Accessed May 3, 2018.

[3] Stern, RS. Prevalence of a history of skin cancer in 2007: results of an incidence-based model. Arch Dermatol 2010; 146(3):279-282.

[4] Guy GP, Machlin SR, Ekwueme DU, Yabroff KR. Prevalence and costs of skin cancer treatment in the U.S., 2002-2006 and 2007-2011. Am J Prev Med 2014; 104(4):e69-e74. doi:dx.doi.org/10.1016/j.amepre.2014.08.036.

[5] R. Siegel and A. Miller, K.D.and Jemal, “Cancer statistics, 2016.,” CA: A Cancer Journal for Clinicians, vol. 66, pp. 7–30, 2016.

[6] K. H, H. Pehamberger, K. Wolf, and M. Binder, “Diagnostic of dermoscopy,” The Lancet Oncology, vol. 3, . 159–165, 2002.

[7] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[8] Litjens, Geert, et al. "A survey on deep learning in medical image analysis." Medical image analysis 42 (2017): 60-88.

[9] Yu, L., Chen, H., Dou, Q., Qin, J., Heng, P.A., 2017a. Automated melanoma recognition in dermoscopy images via very deep residual networks. IEEE Trans. Med.Imaging 36 (4), 994–1004. doi:10.1109/TMI.2016.2642839.

[10] Esteva, A., Kuprel, B., Novoa, R.A., Ko, J., Swetter, S.M., Blau, H.M., Thrun, S.: Dermatologist-level classification of skin cancer with deep neural networks. Nature 542(7639), 115–118 (2017)

[11] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. Rethinking the inception architecture for computer vision. Preprint at https://arxiv.org/abs/1512.00567 (2015).

[12] Russakovsky, O. et al. Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. 115, 211–252 (2015).

[13] Pan, S. J. & Yang, Q. A survey on transfer learning. IEEE Trans. Knowl. Data Eng. 22, 1345–1359 (2010).

[14] Tschandl P., Rosendahl C. & Kittler H. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data 5, 180161 doi.10.1038/sdata.2018.161 (2018)

[15] Noel C. F. Codella, David Gutman, M. Emre Celebi, Brian Helba, Michael A. Marchetti, Stephen W. Dusza, Aadi Kalloo, Konstantinos Liopyris, Nabin Mishra, Harald Kittler, Allan Halpern: “Skin Lesion Analysis Toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI),

Hosted by the International Skin Imaging Collaboration (ISIC)", 2017; arXiv:1710.05006.

[16] Cloud Firestore. (n.d.). Retrieved August 29, 2018, from https://firebase.google.com/docs/firestore/

[17] Chollet, Fran and others, Keras (2018), GitHub repository, https://github.com/keras-team/keras

[18] L. Fei-Fei, R. Fergus and P. Perona. Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. IEEE. CVPR 2004, Workshop on Generative-Model Based Vision. 2004

[19] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," APSIPA Transactions on Signal and Information Processing, vol. 3, p. e2, 2014.

[20] "Documentation for individual models." https://keras.io/applications, 2018. Accessed: 2018-08-01.

[21] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[22] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

[23] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

[24] Drifty, Inc (2016). Ionic. https://ionicframework.com/