BuildingPoint

PROJECT STUDY & IMPLEMENTATION REPORT

David Elimelech, Nicholas Magal & Rafael Marques Braga

DR. SUTCLIFFE & DR. RAMAMOORTHI | UNIVERSITY OF MIAMI - CSC 411

TABLE OF CONTENTS

Introduction ___ ***3***

Problem ___ **3**

The Idea ___ **3**

Specifications ___ ***4***

Compatibility & Design ___ **4**

Features ___ **4**

Target Market ___ **4**

Performance ___ **4**

Requirements ___ ***5***

Background ___ ***6***

Software Stack ___ **6**

Firebase ___ 6
Python ___ 6
TensorFlow ___ 6
Keras ___ 6
Java and Android Studio ___ 6

Machine Learning and Deep Learning for Image Recognition ___ **6**

Neural Networks ___ 7
Convolutional Neural Networks ___ 8

Implementation ___ ***10***

Convolutional Neural Network Design ___ **10**

Dataset Considerations ___ **11**

Geolocation & Confidence ___ **11**

Android ___ **12**

Splash Screen ___ 12
MainActivity ___ 12
GalleryOption ___ 13

Results and Remarks ___ ***14***

Design Elements ___ ***15***

Algorithmic Solution Design ___ **15**

Data Design ___ **16**

Component Design ___ **16**

Deployment Design ____ 17
UI/UX Flow ____ 18
Future Plans ____ *21*
AR Implementation ____ 21
Additional Buildings ____ 21
Increasing Confidence Level ____ 21
Machine Learning Model ____ 21
Machine Learning Model by Geography ____ 21
iOS Version ____ 22

INTRODUCTION

PROBLEM

In today’s world people are consistently navigating through new environments. These people include, but are not limited to, tourists in an exotic new city, freshmen in a new college, and residents in a new city. These people all experience the problem of not being able to identify buildings in their surroundings. Current solutions for this issue are to ask for help or consult Google Maps, these are tedious and non-effective options. The goal of this project will be to develop a system that will easily detect and inform users of what is surrounding them.

THE IDEA

A mobile application will be developed where the device’s sensors and camera will analyze a building and inform the user of its name and relevant information. The building will be identified through a combination of machine learning and geolocation technologies. Once the app has identified the building, it will then display to the user a dialog. The dialog will contain relevant information about the building, such as the name of the building, the businesses within the building, and its address as a link to Google Maps.

SPECIFICATIONS

COMPATIBILITY & DESIGN

The mobile application should be able to support devices using the latest versions of Android (7.0 Nougat – current version (Android 9 – Pie)). It should follow the latest design language which is currently Google's material design.

FEATURES

The mobile application should allow the user to recognize any building by a point and shoot method through a cellphone's camera. The user should also be allowed to choose a picture from the phone's library. The application should show the information about the building in the picture, if recognized. The building's dialog should contain a link to its address in Google Maps.

TARGET MARKET

The target market for this application concept is mainly tourists or residents of a city whom are trying to locate or get more information on a specific building in a city, or campus. These include both males and females, usually aged between the age of 18-50, with a "exploring" focused personality.

PERFORMANCE

The application should be intuitive, easy-to-use, reliable, and fast to respond to a user's interactions. Android updates should not interfere with the performance of the application. An internet connection will be required for the application's features to work as intended.

REQUIREMENTS

1. Create a machine learning model with the ability to correctly classify buildings.

2. Combine user geolocation with machine learning results to improve overall classification prediction.

3. Create a database hosting building information.

4. Synthesize the model into a user-friendly mobile application.

BACKGROUND

SOFTWARE STACK

FIREBASE

Google Firebase is an application development platform that hosts numerous easily accessible features. A custom TensorFlow model can be imported into Firebase's ML Kit, which delivers effortless access to this machine learning model from within mobile applications. Also, Firebase's Firestore, which is a NoSQL document database, allows for the scalable structuring of data, fast queries, offline syncing, strong security, and cloud functions.

PYTHON

Python is a high-level language that excels when combined with big data problems. It has libraries built for these types of problems including: Pandas, NumPy, SciKitLearn, TensorFlow, and Keras. Due to these libraries, Python is an excellent choice for ML development.

TENSORFLOW

TensorFlow is a machine learning package used for creating neural networks. Initially created by engineers from Google's AI organization, it is now an open source software library.[1] Its strength comes from its extensive machine learning packages and is used in many different disciplines.

KERAS

Keras is an intuitive, higher-level API capable of running on top of the TensorFlow machine learning library that allows its users to create powerful neural networks. Due to its relative simplicity compared to TensorFlow, Keras can be used to harvest the power of TensorFlow.

JAVA AND ANDROID STUDIO

Java provides an easy object orientated approach to coding Android applications. Android Studio offers an excellent and concise work environment to code an application in. Android Studio allows auto completion, a diverse selection of virtual devices for debugging, and a user-friendly environment to code in.

MACHINE LEARNING AND DEEP LEARNING FOR IMAGE RECOGNITION

Machine learning (ML) is widely used today to analyze patterns that exist within certain sets of data. ML works to the point that the computer can eventually learn to classify different types of data, and in some use cases, even draw inferences from the data it is introduced to. It was born from the idea that computers can learn from data and use this older data to label and further study newer data. There are many different ML paradigms, each with their own advantages and disadvantages. These include, the k-nearest neighbor classifier, linear & polynomial classifiers, Bayesian classifiers, and neural networks classifiers.

[1] https://www.tensorflow.org

NEURAL NETWORKS

Artificial neural networks are based on the perceptron. The perceptron, which can be thought of as a single layer neural network, is a linear classifier that uses binary classification to predict whether an input belongs to a class or not. When more layers are added to create a multi-layer perceptron, you get what is known as a neural network. Every attribute value that describes a data example is taken as an input signal, and each input signal is attached to each neuron in the next layer using a path that is assigned a certain weight (random to start). Each neuron receives a weighted sum of the input signals and their corresponding weights. That neuron then applies an activation function (sigmoid, rectified linear unit, etc.) to that sum and sends that output to the next layer of neurons as input, repeating the process. The output layer will then contain the probabilities of the input example belonging to each class. Based on the result (probabilities), the error is backpropagated through the network, and the weights of each path are updated. The updating of the weights is what changes the behavior of the network, and that constitutes the act of "learning" in the case of neural networks. [1]

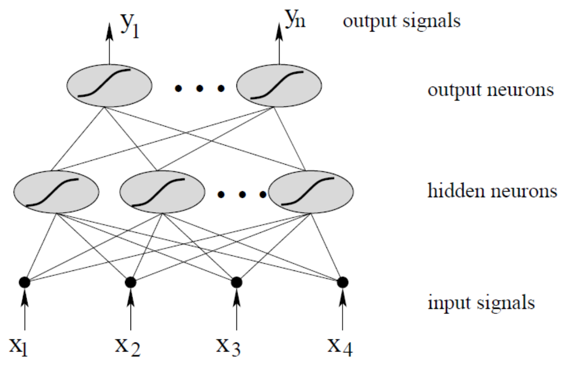
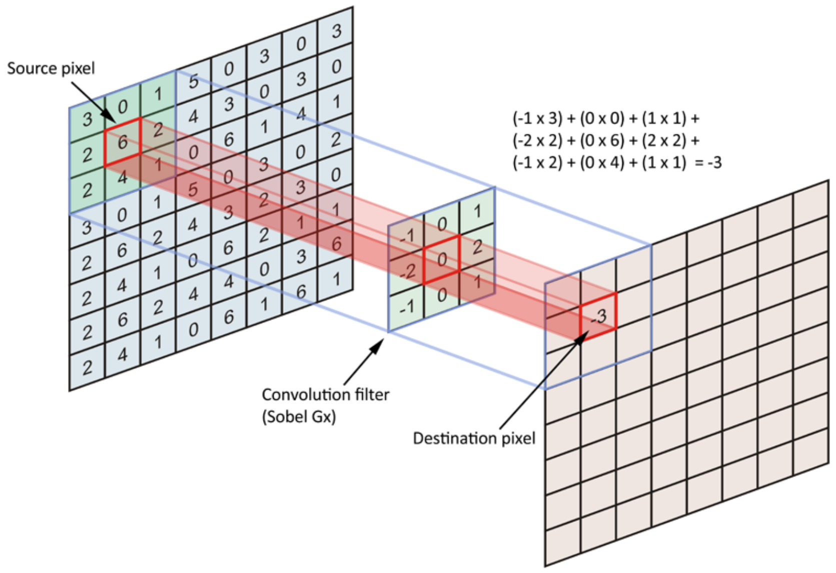


Figure 1 [1]

Figure 2 shows a small representation of how a neural network is built. An important part of neural networks is that each neuron in a layer is connected to all the neurons in the next layer. This is a huge drawback in the use of neural networks since it requires a lot of processing power to properly train the network with a huge number of parameters. When dealing with images, each pixel is an input signal, and so a neural network can easily be overloaded when used for "learning" to classify images. This is where the use of convolutional neural networks come in handy, because their architectures help break down the number of parameters in the network by extracting features from the input image at each consecutive convolutional layer. More about convolutional neural networks will be discussed in the next section.

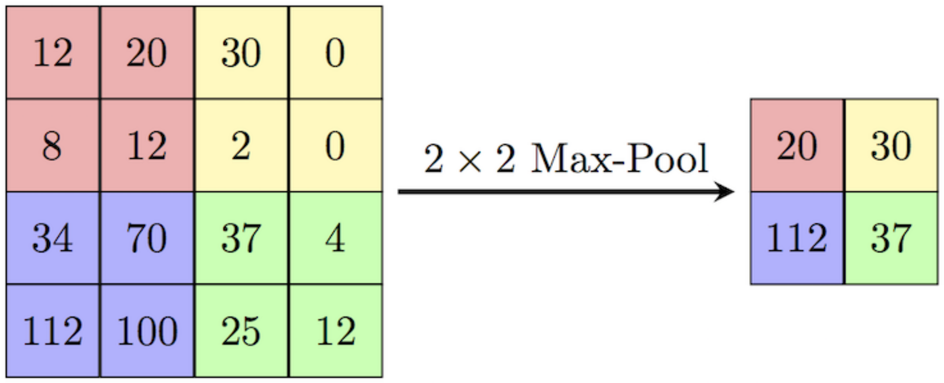
CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (or CNNs for short) are a deep learning algorithm (deep learning because it uses many layers) that excels in image classification. Simply put, CNNs work by taking an input image and breaking it down further by extracting features from the image. This extraction is done using what are known as filters. Filters are matrices that when applied on top of the input image, extract certain features from an image. Some filters are used for edge detection, sharpening, blurring, etc. Each convolutional layer uses many different filters to extract different features from the image, before passing the results to an activation function (often a rectified linear unit to introduce non-linearity because the filters as shown below apply a linear operation).Non-linearity is introduced in order to allow classifiers to model a “response variable (class label in our case) that varies non-linearly with its explanatory variables”.[2] The rectified feature map is then passed to a pooling layer (explained below) before reaching the next convolutional layer. Each consecutive convolutional layer will detect more abstract features than the layer before it.


Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

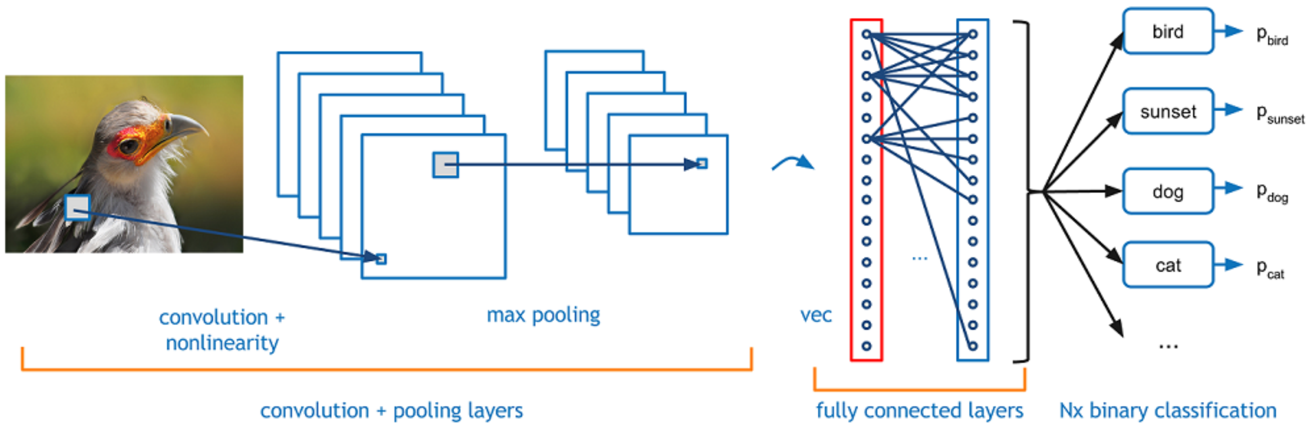
The pooling layer works to reduce dimensionality, which introduces efficiency to CNNs. What the pooling layer does is subsample or retain the most important information from a feature map after going through convolution. The most common type is max pooling, where you define a spatial neighborhood, and allow the max value in that area to represent the entire neighborhood.

[2] https://stackoverflow.com/questions/9782071/why-must-a-nonlinear-activation-function-be-used-in-a-backpropagation-neural-net



By extracting and retaining features as you go along the neural network, CNNs prove to be less computationally intensive than densely connected neural networks in which all inputs are fed to all neurons.

The final part of a CNN is a densely connected neural network, where the "learning" can begin. After breaking down the input image through various convolutional and pooling layers, the image is then flattened (e.g. 10x10 pixel image is now "flattened" into an array of 100 pixels) and each pixel is treated as an input signal to the densely connected neural network. Due to the image processing that took place beforehand, the computer can now learn from the important pieces of information within an image as intended, without having to deal with thousands of parameters. CNNs are essentially densely connected neural networks, with feature extraction layers introduced beforehand to overcome the computational costs incurred by densely connected neural networks alone.



The design of convolutional neural networks is similar to the architecture of neurons in the human brain and was specifically inspired by the organization of the Visual Cortex[3]. A human neuron will only respond to stimuli in a restricted visual field, and the collective overlap of different neurons will provide the image. A convolutional neural network works to replicate this learning process within a computer.

[3] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

IMPLEMENTATION

CONVOLUTIONAL NEURAL NETWORK DESIGN

The first major decision made was whether to train a custom model from scratch or use a method known as transfer learning for creating custom models using an already pre-trained model.

Some benefits in creating a custom model from scratch include the fact that the model can be manipulated and fine-tuned to a dataset, but it comes at a cost. High performing custom models often take days or weeks to train on a personal computer, or even a server. This is where transfer learning comes in, because it can re-use an existing pre-trained machine learning model and re-teach it to fit one's use case. With transfer learning, it often takes under an hour to train a model using a personal computer without a GPU. Also, there are many models to choose from to act as a base for transfer learning. These models are high performing and based on well-known architectures. Thus, using a transfer learning approach, training time can be cut which speeds up the development process. As well, transfer learning can help cut the complexity associated with creating a custom model. It is possible that better performance may be achieved using a transfer learning approach versus a custom model approach.

The way transfer learning works is by exploiting the fact that a CNN learns more basic features (such as edges, shapes, shadows, etc.) in earlier layers, and more abstract features in later layers. Therefore, if given a pre-trained CNN image classifier, it does not have to be retaught how to recognize basic features (early layers), only the more abstract features associated with buildings (later layers). Most CNN's are pre-trained on ImageNet. The ImageNet dataset consists of over 14 million images belonging to over 1000 classes and is considered the benchmark dataset which all new CNN architectures aim to beat, making it the ideal starting point for this project.

The training process would require "freezing" the first few layers of a pre-trained CNN and re-training the model's last couple of layers, using the building dataset. This provides a CNN that correctly classifies buildings while shortcutting the entire process of having to piece together and train a custom model from scratch.

From the base models available for use in transfer learning, a MobileNet architecture was chosen. This is because the MobileNet architecture is high performing, yet small in size. Because the model is lightweight, this allows for the model to be utilized locally by the application rather than over the cloud, speeding up the application's performance.

In constructing the MobileNet, the initial weights were imported from ImageNet. Then, the last 6 layers of the model were discarded, and the final layer of the model was replaced with a softmax layer of 4 output neurons, custom to the current implementation (4 buildings from the University of Miami). After which, all but the last 23 layers of the model (the 23 layers included all of the densely-connected and part of the convolutional layers) were frozen prior to training.

After training, this model was then saved into a .tflite file and imported into Firebase's ML Kit as a custom model.

DATASET CONSIDERATIONS

The dataset needs to be just as good as the model, and in order to avoid overfitting (model trains well but does not test well), it must be ensured that there is a bit of variance/noise in the data. From this, the model will learn to generalize instead of looking for dead-on patterns to declare an image belonging to a certain class. Thus, at least 200 photos of each building were taken from varying angles. These photos were then reviewed in order to remove photos that were too similar to each other, considerably reducing the dataset. After the photos were reviewed, a total of 130 photos were left for each class to be used as input. From there, data augmentations were applied to these photos. Data augmentation takes a photo, applies several transformations to that photo (zoom, blur, rotate, etc.), and offers an even larger and diverse data set. This is critical, because it cannot be assumed that the user is going to take the "best" photo every time; the model needs some flexibility.

In preparation for training, the data was resized to 224x224, shuffled, normalized, one-hot encoded, and split into 80% train and 20% test sets. One-hot encoding is a form of labeling that neural nets use to label classes. In one-hot encoding a label array is created for each image with each column representing a class. The column that corresponds to the class is given the value as 1, all others are 0. For example, for a photo belonging to the first class in a domain of 4 classes would be labeled as [1,0,0,0].

GEOLOCATION & CONFIDENCE

While one of the options was to insert geolocation data as input into the CNN at the densely connected part, it was decided that this would not be the best implementation. This is mainly because the design would deviate too far away from the well-known CNN architectures. By sticking to a well-defined architecture, less experimentation on the network parameters (layers, filters, dropout rates, etc.) is required in order to get adequate results, thus, speeding up development time.

Instead, the geolocation data is used as a validation step in the classification chain. Thus, the geolocation is not used to influence the CNN model's decision, but rather to double-check the model's output. If the user is at least 150m away from the most probable building returned by the model, the user is alerted of background noise, which indicates the model does not have the image captured in its domain. Also, regardless of geolocation, if the probability associated with the most probable building returned by the model is less than 70%, the user is alerted of background noise as well.

ANDROID

The app can be divided into three sections-the Splash Screen, the MainActivity, and the GalleryOption (gallery). Each of these sections corresponds to their own unique activity in the Android application.

SPLASH SCREEN

The splash screen is the welcome screen for the app, and it is displayed for 3 seconds upon the launch of the app. This displays both the BuildingPoint and University of Miami logo with a green and orange theme.

MAINACTIVITY

The MainActivity hosts the main UI, which is a camera preview that when clicked on, will display a dialog that describes the result of the image classification. The app will not work unless the user grants permission for GPS and camera privileges, which is requested when the application is initially launched.

The camera package is used for use of the camera on the hardware. The camera package provides an easy and intuitive way to grab a smartphone's camera and use it in the app. In order to use the camera, a method was created to cycle through all available cameras. After cycling through all available cameras, the ID of the first available back facing camera is saved. This is used to open the camera.

A texture view is used to preview what the camera is looking at. To create an appealing UI, the texture view is formatted to span the entire screen. This texture view hosts the camera preview.

Once the user clicks on the screen, a photo is captured alongside a geolocation call. The photo is captured in bytes and is then converted into a bitmap in order to be used by the machine learning model. It is important to note that although a photo is captured, it is never saved. Once the photo passes through the model it is not used anymore and is not saved onto the smartphone's memory.

A geolocation call consists of using the Fused Location Provider. The Fused Location Provider is an API provided by Google Services that intelligently combines different signals to provide the location information of your phone[4]. After calling on the ML model for inference, the app will request for the last known location of the user by using the fore mentioned geolocation call. This location will then be utilized to calculate the distance between the user and each building, providing some context surrounding the ML model's prediction.

Using both the ML model's prediction and the geolocation data, the app will display to the user a dialog containing the classified building's information if the classification was successful. If the classification was not successful a background noise dialog is displayed to the user.

As soon as the user clicks the texture view, the camera's preview is frozen on what it was previewing before it was clicked. This allows the user to be able to preview the photo that they have sent to the classification algorithm.

After the photo has passed through the classification algorithm (implementation details above), a dialog is then created that displays the information of the classified building. The dialog is a standard dialog that has been customized to fit to the bottom of the screen that includes a slide in animation as well as a green and orange color

[4] http://www.cs.miami.edu/home/geoff/Courses/CSC330-18F/Content/SensorsLocators.html

scheme. The address contained on this dialog is also interactive and will send the user to the Google Maps application with the queried address when clicked.

GALLERYOPTION

The next section, the GalleryOption, is very similar to the MainActivity. This activity allows a user to choose a photo from their gallery for classification. The only difference with this activity and the MainActivity is instead of a camera instance and texture view being used to show what the user is previewing, an ImageView is used to host the image chosen from the gallery to be classified. This image that is hosted in the ImageView is also the image passed to the classification model.

RESULTS AND REMARKS

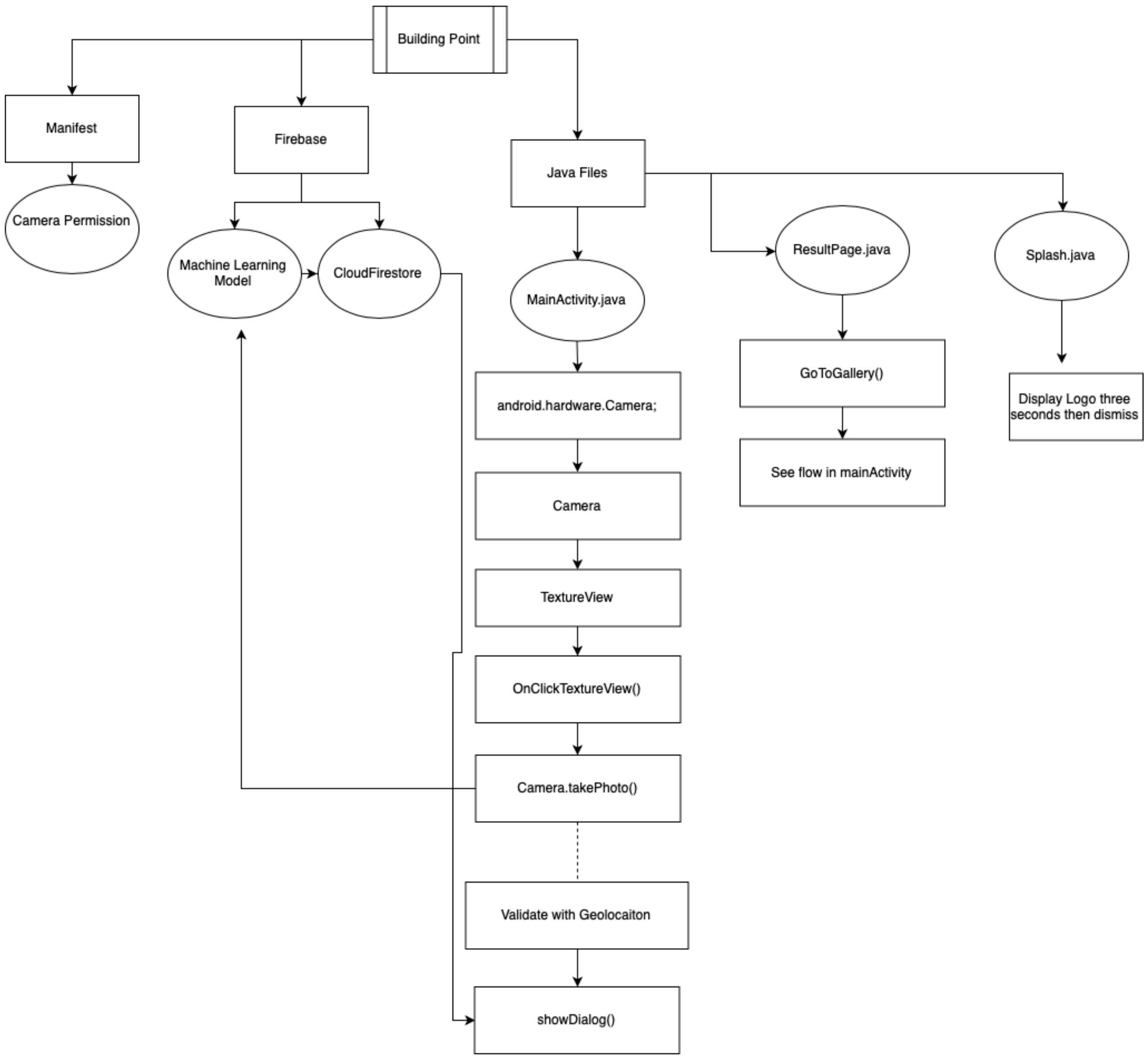
After gathering and cleaning the data, pre-processing the data for input, building the CNN model, and training the model using a transfer learning approach, the model was able to achieve over 90% accuracy on both the testing and training sets. Altogether, training the model took 10 epochs and was completed in under 20 minutes. And although there was some initial overfitting while training, this was combatted using data augmentation, cutting epochs, increasing data shuffling, reducing the complexity of the model (preprocessing functions), and adding more images.

After implementing the MobileNet architecture, the model came out to be just under 13 MB. Relative to other machine learning models, this is a pretty small size. This small size allowed the model to be stored locally on the app which allowed for the increased speed and performance of BuildingPoint. Now internet connection is only needed for access to the Firestore database for queries.

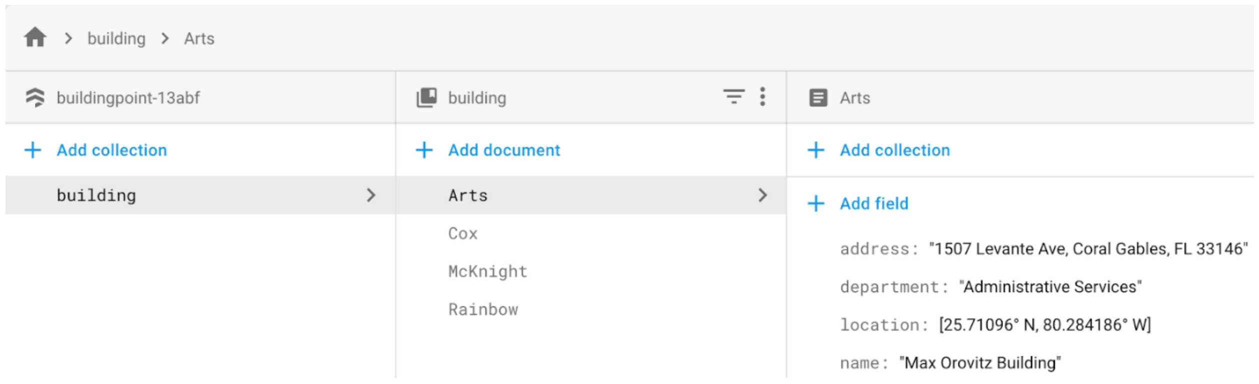
Finally, the model was consolidated into a user-friendly Android application. Although many bugs were encountered while building the app such as the camera crashing and sluggish load times, all major bugs were eventually overcome for the alpha release of the application. Strategies such as cutting down the number of database queries, carefully releasing the camera and its preview, and adding activity control variables helped add to the increased robustness of the app.

DESIGN ELEMENTS

ALGORITHMIC SOLUTION DESIGN



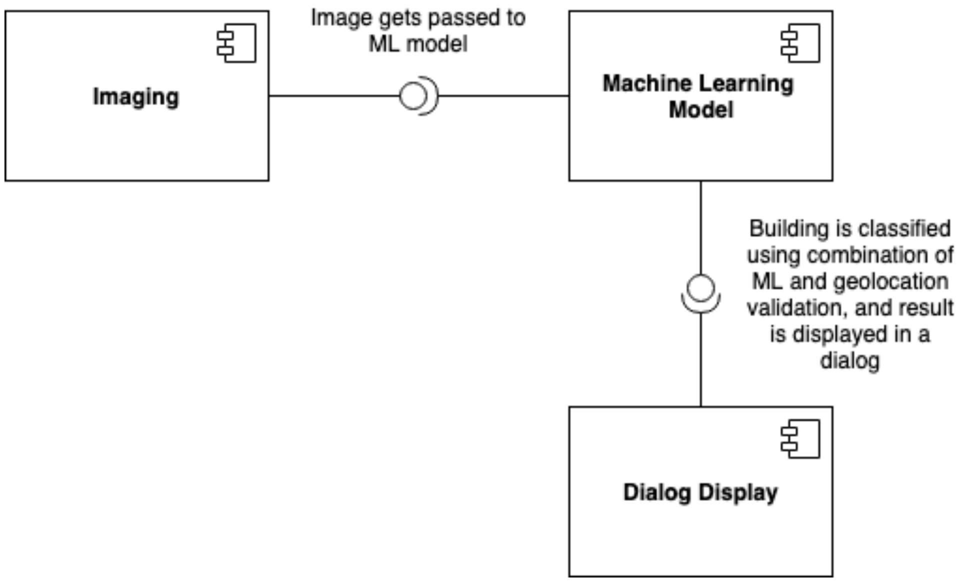
DATA DESIGN


building > Arts

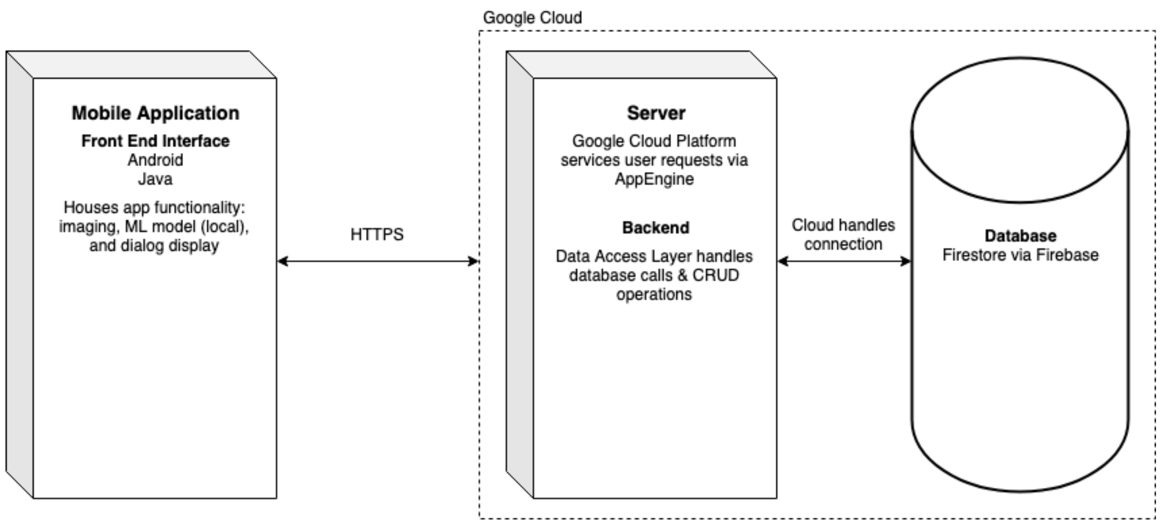
buildingpoint-13abf	building	Arts
Add collection	Add document	Add collection
building	Arts	Add field
	Cox	address: "1507 Levante Ave, Coral Gables, FL 33146"
	McKnight	department: "Administrative Services"
	Rainbow	location: [25.71096° N, 80.284186° W]
		name: "Max Orovitz Building"

Above is Firestore. The information about each building is stored in a document, and each of those documents is stored in a collection.

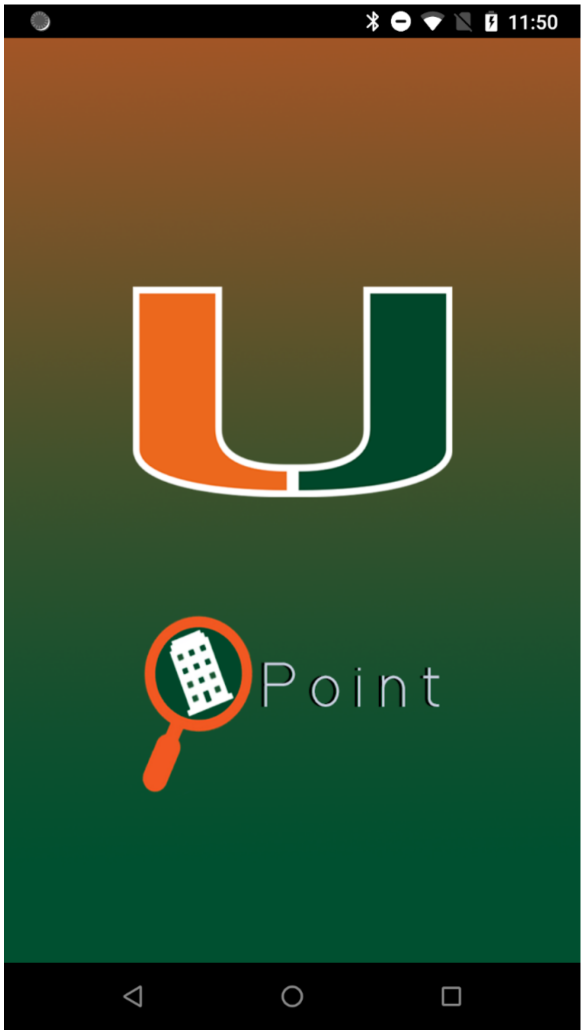
COMPONENT DESIGN


Imaging
Image gets passed to ML model
Machine Learning Model
Building is classified using combination of ML and geolocation validation, and result is displayed in a dialog
Dialog Display

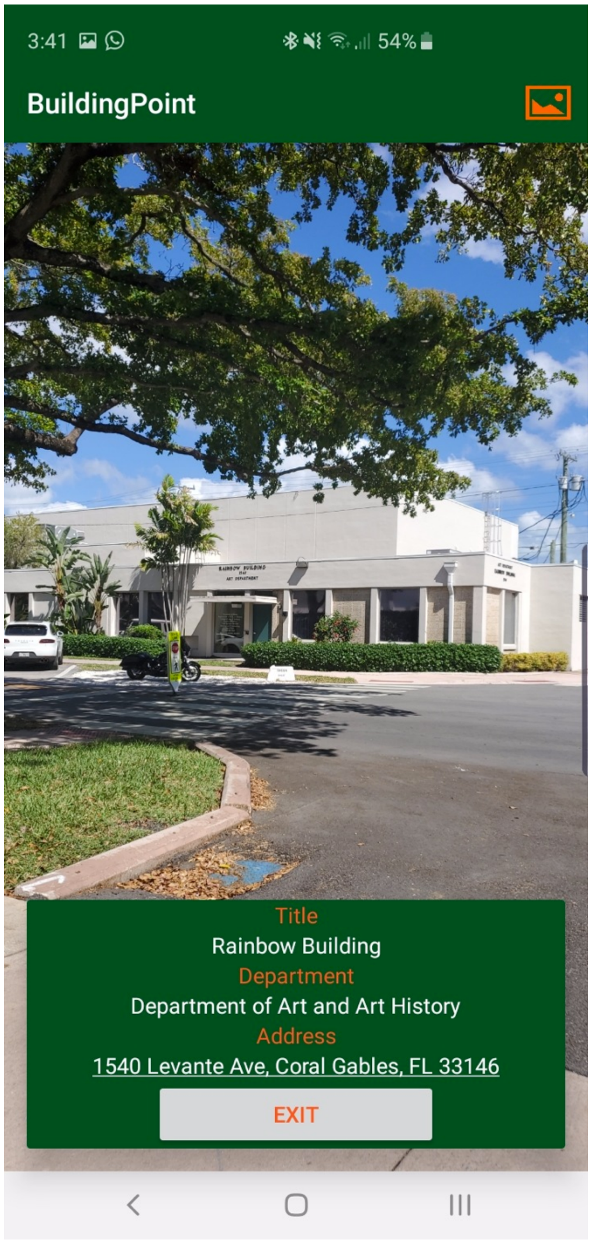
DEPLOYMENT DESIGN


Google Cloud
Mobile Application
Front End Interface
Android
Java
Houses app functionality: imaging, ML model (local), and dialog display
HTTPS
Server
Google Cloud Platform services user requests via AppEngine
Backend
Data Access Layer handles database calls & CRUD operations
Cloud handles connection
Database
Firestore via Firebase

UI/UX FLOW



The Splash Screen is displayed for 3 seconds on launch.



Title

Rainbow Building

Department

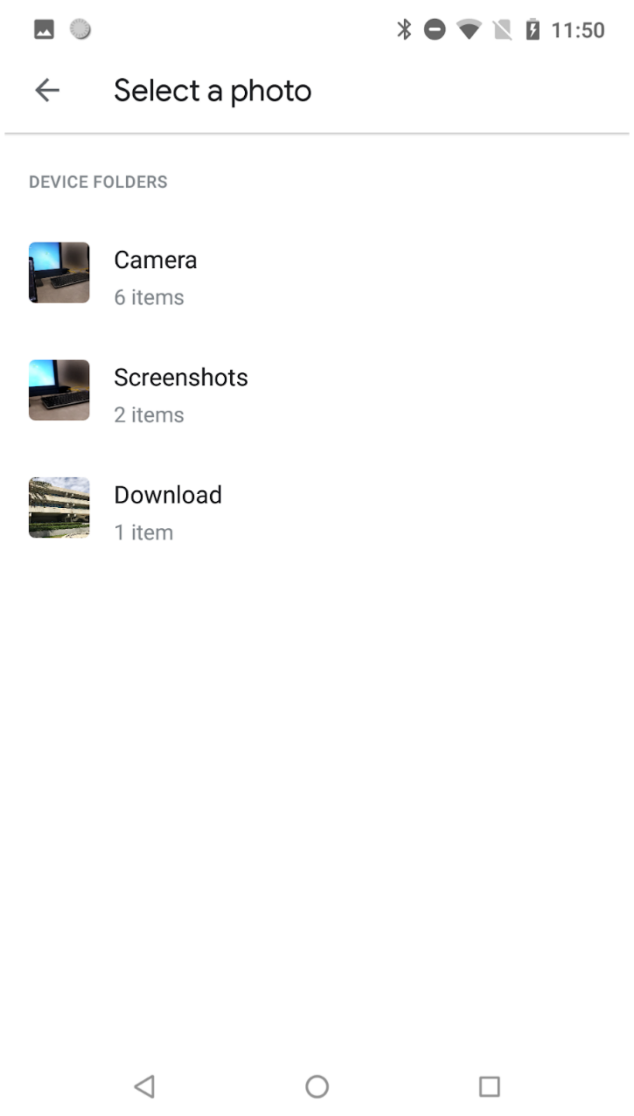
Department of Art and Art History

Address

1540 Levante Ave, Coral Gables, FL 33146

The MainActivity Page. On this page the user can click the screen to capture a photo. Then, if the photo is classified successfully, a dialog pops up with relevant information about the classified building. Else, if unsuccessful, a background noise dialog is shown. Among the information shown on a building's dialog is a link to their address in Google Maps, which will open up a Google Map's query when clicked.

In the top right corner there is a button that takes the user to photos saved in their gallery, which gives them the option to classify a photo stored in their gallery.



The GalleryOption Page. Secondary page of the app which takes user to the gallery. After selecting a photo from the gallery, if classification was successful, a dialog pops up on top of the selected image (similar to the MainActivity Page). From there, the user can select to either go back to gallery or back to the camera (MainActivity).

FUTURE PLANS

AR IMPLEMENTATION

In order to provide a more intuitive and interactive experience, Augmented Reality could be implemented into future versions of BuildingPoint. AR would be used to mark which building has been scanned and would be implemented as an outline superimposed over the building scanned.

ADDITIONAL BUILDINGS

Currently, only 4 buildings are in the model. Not only are there plans to add every building on campus into the model, but also there are plans to expand the model by including buildings in major cities. Possible city locations include New York, Miami, etc.

If it is decided on adding major cities, the current method of collecting data would not be sufficient (taking 100s of photos of 100s of buildings). In order to bypass this issue, two solutions would be used - use an API that can gather the data from online sources such as Bing Images, or introduce a program to incentivize select users to gather data for the machine learning model.

INCREASING CONFIDENCE LEVEL

In order to make the classifier more accurate, there are several improvements to the current algorithm that could be implemented. Some options that are currently being evaluated are: increase the closest buildings probability by a certain increment, checking the 3 most probable buildings from the machine learning algorithm and comparing their distance to the user's current location, and adding an algorithm that takes into account the user's direction, weather, and time of the day.

MACHINE LEARNING MODEL

In the future, another possibility would be to create a custom model, or, even trade in the lightweight MobileNet for a slightly more heavyweight architecture (more data, more accuracy) to perform transfer learning on. With a heavyweight CNN, there may be the need to lean on using cloud inference versus local requests, but this is of course subject to future considerations. Not to mention, even with the existing model, there is a lot of fine-tuning that can be done (hyperparameter optimization) in terms of the amount of data, processing of the data, layers of the model, training of the model, etc. As well, regarding geolocation, there is much to consider in terms of how to best utilize the geolocation data. Whether that be integrating it into the model as highlighted earlier, considering other variables like orientation, etc. Nevertheless, as the field of machine learning continues to evolve, considerations as to how to best implement BuildingPoint will continue to evolve as well.

MACHINE LEARNING MODEL BY GEOGRAPHY

Once BuildingPoint is implemented on a larger scale, the domain of buildings will be vast. The more domains in a machine learning model, the more complex the model becomes and the longer it takes to classify. It is therefore important that separate machine learning models are created for each geographic region to avoid overloading the model with too large of a domain set. For example, in the Miami area, there would be a separate models for Brickell, the University of Miami, South Beach, etc.

Though an important note to consider is that this method may be unnecessary if we implement a geolocation filter on the classifier. If we can have only certain parts of the classifier activated based on the location of the user, the issue of a running a large machine learning model will be overcome.

IOS VERSION

In the USA, Apple has a 43% market share of mobile applications[5]. Thus, going forward, it is necessary to eventually create an iOS app using Swift or possibly Flutter. The iOS app will perform the same as the Android app.

[5] https://www.statista.com/statistics/266572/market-share-held-by-smartphone-platforms-in-the-united-states/