The Virtual Field Trip Scavenger Hunt:

An application of various image recognition techniques to enhance the virtual field trip experience.

Adam Carter and Bill Capps
School of Computing and Engineering
University of Missouri, Kansas City
Kansas City, USA
arcbpc@mail.umkc.edu, bjctgc@mail.umkc.edu

***Abstract*— As VR applications becomes more prevalent and inexpensive, numerous opportunities exist to apply these advances in an education setting. One way to enhance the education experience is to add a level of intelligence and interaction to the basic VR experience. For our project, we propose to use a variety of machine and deep learning techniques to allows students to interrogate their environment through image recognition. The framework for this interaction is a virtual scavenger hunt, where students interrogate their environment looking for specific items. We use image detection approaches based on multiple learning engines in order to improve engagement in the VR experience, which has the potential to improve learning outcomes.**

Keywords—education, immersive, virtual reality, machine learning, artificial intelligence, interactive, Spark, Tensorflow, Android, Google cardboard

I. INTRODUCTION

For several decades, advancements have been made in image recognition and analysis methods. Machine learning is one such approach that focuses on training a system to improve in its ability to recognize patterns and classify images. Machine learning researcher Tom M Mitchell provides a definition of machine learning thusly [1]:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

There are multiple learning approaches, generally divided into supervised and unsupervised learning strategies. In supervised learning, a system is provided with a suitably robust data set of training data which is used to teach the system. Training data usually consists of data elements and labels describing those elements, with the goal of the system to generate predictions for new data within the realm of known labels, also known as classifiers. A separate set of test data that was not included in the training data is used to validate the effectiveness of the model. There are a variety of mathematical models and architectural approaches to facilitate and iteratively improve the performance of such models, some of which will be discussed later in this paper.

Unsupervised learning consists of using historic data to derive conclusions perhaps previously unknown about large volumes of data. Examples of unsupervised learning could be to determine whether patterns exist in stock pricing based on historical trends. By analyzing a clustering data, we discover patterns that may not otherwise emerge in the data.

The goal of image recognition is to eventually achieve consistent unsupervised learning results, but the reality is that supervised learning approaches are currently more effective and easier to train. One goal of our project is to explore the viability of several different unsupervised learning approaches to compare performance and application to our image analysis and prediction problem. To that end, we intend to compare the performance and accuracy of three different systems: The Clarifai open API [2], Spark MLib [3], and Tensorflow [4]. Each of these systems uses a different approach, though we will largely focus for the rest of the paper on the implementation and comparison between Spark MLib approach and Tensorflow.

Another interesting domain is the rise of VR applications over the past few years. The commercial viability of products like the Oculus Rift [5], as well as low-end offerings like Google Cardboard [6], have made VR a much more accessible experience. Many VR applications are currently static interactions: A 360° view or panorama is presented, and users can look all around their environment. More expansive VR experiences add some interaction with animated or programmatic elements that users can interact with via external controls. This creates a challenge for developers, since the immersive experience requires the participants to feel they are part of their environment, a concept known as presence, which is difficult even for many top end applications to provide.

Most applications, particularly in the education space, stop at simply being able to render a panoramic view. This is partially because rendering high definition views that are fully immersive requires expensive equipment capable of capturing images or video in that format, but also due to the high technical requirements to develop software in this space. Smaller applications like the Google Cardboard Photo, allow users to create panoramic images that can be rendered into VR images, though there are often large portions of the top and bottom zones in the field of vision that cannot be captured with a simple phone-based camera.

Our application will feature a Google Cardboard application that will feature the standard non-dynamic

panoramic view, but to that system we intend to add a layer of interaction not found in other educational VR applications. Specifically, our goal is to create a scavenger hunt experience, where students may progress through a series of locations and are assigned an item to find in that location. Rather than rely on static object collision testing commonly featured in video games, we intend to use image recognition techniques to allow the user to interrogate their environment, using questions such as "What am I looking at?" and "Did I find it?" to allow them to discover and experience these locations in a way that simply looking at them could not provide.

Due to the limitations on time and experience in this domain, this project aims to provide a proof of concept to this approach rather than a fully realized application that spans multiple venues. We believe that the techniques we apply here could easily be expanded to encompass multiple dynamic environments, each with its own unique set of highly accurate predictive models. This would make such an application highly dynamic and simple to add new content to.

II. Related Work

The topic of image recognition is one that has been under study for multiple decades. It is not the intent of this paper to provide a background in that topic. This paper presupposes the reader has a basic understanding of machine learning approaches and models. We will site specific models or approaches as they become relevant to our project.

We similarly presuppose the reader has at least a passing understanding of VR technology. References have already been provided to the Google Cardboard approach we will be using in this project, though plenty of similar work has been done with other implementations such as the Oculus Rift or Samsung Gear VR. This project does not intend to improve or iterate over those frameworks; rather we intend to adapt both approaches to suit our particular problem space.

We will present some limited related findings as regards the usage of VR, particularly the idea of virtual field trips, in the education domain so far. Perhaps the most notable initiative in this regard is Google Expeditions [8]. The project, designed specifically to be easy to adapt for low income areas using the Google Cardboard headset and released in the Summer of 2016, is a collection of interesting locations shot in high definition 360° imagery. These provide a variety of interesting education settings for students to explore. While the images are static and non-interactive, other larger tools exist for educators to monitor what students are looking at to provide instructions or explanations for objects rendered in those virtual locations.

Clay Middle School in Kansas has been forming their own unique library of virtual field trip imagery [9]. They have a collection of web pages that offer virtual tours of places with educational value. NYT VR [10] takes users to places of significance and includes information about the associated New York Times stories. Titans of Space [11] lets users explore the solar system using virtual reality. More recently, a startup company called The V Form Alliance [12] announced they were seeking funding to pursue a relevant project. Their stated mission is to "allow elementary and middle school students to take a virtual reality 'field trip' exploring landmarks in Kansas and Missouri that are relevant to black history."

One similarity of many of these virtual tour or field trip applications is that they are essentially non-interactive. They present in many cases beautifully rendered panoramas that are interesting to explore, but give no framework for interacting or interrogating their environment. The panorama approach gives a very low barrier to entry for such initiatives, but the lack of interaction causes the experience to become static and predictable after multiple views. This is the primary problem we wish to address with our project.

III. Proposed Work

The overall goal of this project is to create an interactive application that uses a question and answer approach to interact with a virtual environment. In order to do this, we designed a system with the following distinct components:

- Google Cardboard App for Android
- Tensorflow Model for Image Recognition
- Spark MLib Model for Image Recognition
- Conversation API for Question and Answer interactions

Our system is designed to have the following component architecture:

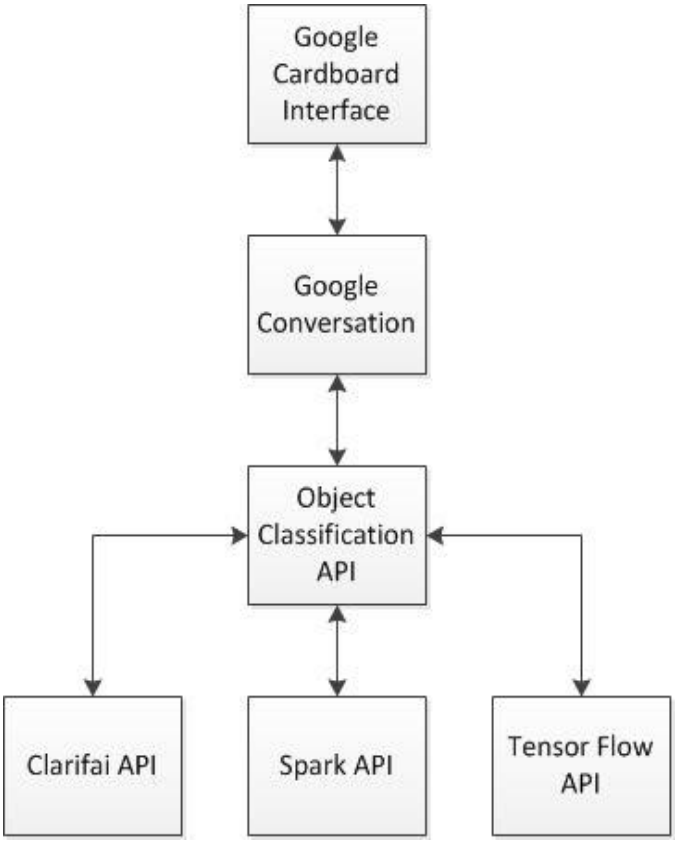

Fig. 1. Component Architecture

A secondary but significant objective of this project is to provide a comparison between the various image recognition approaches, comparing a public API (Clarifai) to a Machine Learning model (Spark MLib) and to a Deep Learning model (Tensorflow).

While we cannot control the input and training elements for the Clarifai model, we were able to devise our own training and test data sets for Spark and Tensorflow. We believe this provides a fairly robust framework for comparison between the two approaches, given that we expect the Clarifai API to be the least accurate and poorly suited to our problem.

To implement our project, we needed a testing data set that was controllable and easy to build out. To do this, we began with a panoramic view of a game room. There are nine classifiers in our dataset, corresponding to key visual elements in the room. Each element is unique, though several share common characteristics. For example, there are four posters in our classifier set, each consisting of a range of colors and being approximately the same dimensions in real life. We believe this gives us a use case similar to many we might extend this model to, such as similar styles of paintings in a museum, or multiple statues in the same image.

From this dataset, we chose our approaches for each model. For Spark, we use the AKAZE [13] method for feature detection in images. This approach tends to yield better runtime performance than SIFT [14], an alternative approach covered in many tutorial methods, but requires a larger dataset to be accurate. We train our model using a Random Forest Decision Tree approach [15]. For Tensforflow, we use a Convolutional Neural Network [16] consisting of two layers. The last layer in the model is a Softmax function [17] designed to help us finalize our results by classifier.

To build our dataset, we took the game room panoramic image which was captured using Google Cardboard Camera [18]. One realization we had from our development effort is that the domain of possible image captures that can be taken from our VR panorama is fixed. Thus, rather than use indirect images or data from a different source, we decided to sample the core VR image from which the snapshots will be generated. To do this, we defined a viewable box around each of our core visual elements that ensured that at least 50% of the desired element was included in the image, then drew a number of 400 x 400 pixel subimages that represent the item to be classified. This resulted in between 10k – 15k images per classifier. From these images, we then sampled a collection of 150 images per classifier to form a new dataset drawn from the VR image. We then further sampled another 10 images per classified to form our validation data set.

IV. Approach Algorithms

While there is no algorithm to speak of with Clarifai, since that is an open API and one we do not contribute to, we can present a brief overview of the coding algorithms used by our code. This overview will only give a broad overview of features. The code itself is not large enough in scope that it would prove difficult for the reader to understand, and is sufficiently commented and modularized to ease understanding.

First, we will discuss our Spark MLib approach. From a high level, the following steps will be performed:

- Initialize the Spark Instance
- Load our Training Data Set, consisting of 9 classes
- Create Image Descriptions for our images using the AKAZE model for feature detection.
    - Store the corresponding Histograms in Spark
- Execute KMeans Clustering against the feature Histograms.
    - Cluster to 400 clusters using 25 iterations
    - Store the corresponding Clusters in Spark
- Generate Bag of Words Histograms based on the resultant clusters.
    - Store the Resulting Histograms in Spark
- Use a Random Forest model to train our model
    - Random Forest Generation uses the following criteria
        - Split data 80/20% for training/testing.
        - Generate random trees of verying depth and using multiple impurity approaches.
        - Compare each generated tree result looking for the best outcomes
    - Store the best approach in Spark
- Process Validation/Testing data to compare model performance
    - Generate a feature histogram for each image
    - Process Image through Model
    - Score Results and Build Confusion Matrix

Here is some sample code used in generating the Random Forest Model, which is the key element of this model generation (Written in Scala):

```
\
val maxBins = 100
val numOfTrees = 4 to(10, 1)
val strategies = List("all", "sqrt", "log2", "onethird")
val maxDepths = 3 to(6, 1)
val impurities = List("gini", "entropy")

var bestModel: Option[RandomForestModel] = None
var bestErr = 1.0
val bestParams = new mutable.HashMap[Any, Any]()
var bestnumTrees = 0
var bestFeatureSubSet = ""
var bestimpurity = ""
var bestmaxdepth = 0

numOfTrees.foreach(numTrees => {
 strategies.foreach(featureSubsetStrategy => {
  impurities.foreach(impurity => {
   maxDepths.foreach(maxDepth => {

    println("numTrees " + numTrees + " featureSubsetStrategy " + featureSubsetStrategy +
     " impurity " + impurity + " maxDepth " + maxDepth)

    val model = RandomForest.trainClassifier(training, numClasses, categoricalFeaturesInfo,
     numTrees, featureSubsetStrategy, impurity, maxDepth, maxBins)

    val predictionAndLabel = test.map { point =>
     val prediction = model.predict(point.features)
     (point.label, prediction)
    }

    val testErr = predictionAndLabel.filter(r => r._1 != r._2).count.toDouble / test.count()
    println("Test Error = " + testErr)
    ModelEvaluation.evaluateModel(predictionAndLabel)
```

Fig. 2. Random forest code sample

The Tensorflow Approach follows the basic CNN approach:

- Convolutional Layer
- Rectified Linear Unit (ReLU) Layer
- Max Pooling Layer

Specifically, we add two layers that follow the above approach, compacting our data using Pooling layers as we go. We then allow for node dropout to avoid overfitting, and conclude our model with a softmax layer that generates a resulting Tensor of size 9, matching our 9 image classifiers. Presented below is sample code performing the significant work of our neural network.

```
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])
x_image = tf.reshape(x, [-1, 200, 200, 1])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

#Because we are pooling twice, our feature list is 1/4 the size * 64 for our biases
#So in my case, 200 / 4 = 50, so the tensor needs to be multiples of 50 * 50 * 64
W_fc1 = weight_variable([50 * 50 * 64, 1024])
b_fc1 = bias_variable([1024])

#Need the same new pixel size here.
h_pool2_flat = tf.reshape(h_pool2, [-1, 50 * 50 * 64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
# Dropout
keep_prob = tf.placeholder(tf.float32, name='keep_prob')
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob, name='drop_out')
# readout layer
# Need to change to match number of classes, so 9 for both
W_fc2 = weight_variable([1024, 9])
b_fc2 = bias_variable([9])

y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
y_softmax = tf.nn.softmax_cross_entropy_with_logits(logits=y_conv, labels=y_, name='y_softmax')
cross_entropy = tf.reduce_mean(y_softmax)
cross_entropy_mean = tf.reduce_mean(cross_entropy, name='cross_entropy')

train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

# Needed for graph writing below
values, indices = tf.nn.top_k(y_softmax, 9)
prediction_classes = tf.contrib.lookup.index_to_string(
   tf.to_int64(indices), mapping=tf.constant(labelMaster))
```

Fig. 3. Neural network code sample

V. Implementation & Evaluation

To implement this project, we use the following technologies:

- Google Cardboard App for Android (Java using Android Studio)
- Tensorflow Model for Image Recognition (Python and Flask)
- Spark MLib Model for Image Recognition (Spark, Scala)
- Conversation API for Question and Answer interactions

Because our system is decomposed and implemented in different languages per component, there is no meaningful class diagram or model, aside from the component model provided in Section 3, that would be helpful in describing the system. We will discuss the various implementations for each architectural component.

A. Training Data

Presented are some of the screen shots taken from our application. We'll give a quick overview of some of the visual artifacts used to build our dataset, then some of the artifacts created by our application.

Fig. 4. Basic VR Image of Game Room used as the basic for early analysis

Fig. 5. Screenshot of Google Cardboard Viewer while running

Fig. 6. Six images captured from our Android Application of different areas in the VR Viewer

B. Android Application

This application leverages the VrPanoramaView implemented by Google VR. This is separate from the new GvrView full VR implementation using OpenGL, which means it has limited functionality, but is sufficient for this projects' needs. This means that the native onCardboardTrigger() method cannot be used, so we instead trap any screen touch event (which is essentially what the trigger does) to take a screen shot using the Pitch and Yaw recorded by the headset to determine the current center of the Panoramic Image.

One difficulty of this task was handling images that would wrap around the 'seam' of the image, where the flat image boundaries would need to be crossed to complete the screenshot. Because the VrPanoramaView is more limited in its feature set, the native Android capability of capturing a screenshot is not available. A custom algorithm had to be written to find the current view center and grab the screen shot manually. For the case where we would wrap the scene, a separate image file is referenced that has been rotated 180° so we can cleanly capture the seam.

The image recognition services were created as RESTful APIs. This gives the benefit of creating a separation of the image recognition code from the Android application. Allowing the image recognition code to be run separate from the user's device means that a more powerful machine can do the image analysis, the bulky models do not need to be distributed to the handheld devices, and the models can be updated without a need for a user to update the application. However, this also means that users are required to maintain an internet connection, image data will be transferred on potentially metered connections, and we have to maintain servers hosting the APIs or the application cannot function. As a commercial application, this means that our target audience would need to primarily use Wi-Fi and be willing to pay a monthly subscription fee.

C. Spark Image Recognition

The tutorial code we were provided initially had to be uplifted because OS-specific errors were occurring that stem from an older version of OpenCV. To overcome this, the JavaCP and JavaCPP libraries had to be uplifted from version 0.11 to 1.3.1. This also meant that features like SIFT Feature Extraction were no longer available (as they are not freely available for patent issues). The code was converted to use the AKAZE feature extraction model, and the results were faster performance wise, but lead to smaller feature vectors and less accurate prediction. Results are descripted below in the documentation setting.

While the performance of the system is significantly improved, there are still several current limits to this approach. First are the OS issues that prevent using a larger dataset (OpenCV is not entirely stable, and frequently causes fatal OS errors that interrupt parsing). We were able to generate a more consistent dataset that was sufficiently large in scope to get good accuracy, but that increase caps not much further beyond where we drew the line. Second, I believe converting to Greyscale significantly reduces accuracy. Both SIFT and AKAZE models tend to strongly favor one image (The Star Wars posters, which are chosen significantly more frequently than any other classifier) which would appear different if processing RGB vectors separately. That change was determined to be outside of a realistic scope for this iteration. Finally, the low quality of our VR screenshots in particular makes accurate analysis more challenging. Many of these shortcomings could be addressed with better source imaging and better hardware to process the model.

D. Tensorflow Image Recognition

Using the same dataset as the Spark MLib approach, the Tensorflow approach had to actually compress the image in order to process successfully. Specifically, we had to compress the image size from 400 x 400 to 200 x 200. By doing that, we were able to successfully build a multilayer CNN that was able to process our training data, then evaluate itself against our test data. This process took a tremendous amount of time to compete, however. It took approximately two hours to complete 120 iterations through our CNN, with a random data sampling of 10%, or 135 images, each iteration. Further discussion of the results of this approach are shown below.

E. Conversation API

This application does not readily lend itself to a natural conversation. Variations of the application, such as a virtual field trip, would be better suited to conversation and allow children to ask questions similar to those that they would ask a tour guide or teacher. For our scavenger hunt software, the user side of the conversation is mostly limited to asking if the desired object was found. There really isn't a need for the user to vocalize this question since it is implied by pressing the only available button. The application side of the conversation consists of stating an object to be found and validating the user's answer. Due to the project requirements, we use a conversation API to facilitate the dialogue, but in reality, these conversations are primitive and would be much simpler to implement directly in the Android application. This is because there is no need for natural language processing to interpret what the user is requesting.

F. Training Results

In general, both generated models are highly accurate. With the Spark AKAZE model, we were able to achieve 94.4% accuracy. With the Tensorflow model, we were able to achieve a full 100% accuracy for the same test data. An interesting distinction between the two is that the Tensorflow CNN approach handles 'sparse' images better than the AKAZE feature detection approach. By sparse image, we mean an image with very few defining details; large blocks of single color patches, very few edges. An example image might be a national flag. While the CNN approach used by tensorflow can fairly accurately learn colors and match them to labels, AKAZE struggles because flags feature a very low number of visual edges. This problem is compounded by reducing the data to greyscale. We found that the AKAZE mode, while highly accurate, still struggles with one particular object, which was a table with a large blue surface. In many potential images, there are very few objects in the background as well, making it much harder for AKAZE to identify this particular category. For this table object, testing accuracy was only 60%, while nearly every other classifier achieved 100%. This can be observed in the provided Confusion Matrix:

```
|======== Confusion matrix ==========
10.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0 10.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0 10.0  0.0  0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0 10.0  0.0  0.0  0.0  0.0  0.0
 0.0  1.0  0.0  0.0  9.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0 10.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0 10.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  6.0  4.0
 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 10.0
0.9444444444444444
```

Fig. 7. Confusion matrix

For Tensorflow, we probably ran more iterations than was strictly required, but the random weight and bias assignment at the beginning lead to less accurate outcomes occasionally when only run through 100 iterations. We ran 120 executions with a 10% sampling (135 images per iteration). We hit a perfect 100% accuracy by the time we hit iteration 100 and never dropped from there. Here is the logging from the execution of the Tensorflow job:

```
Number of Classes: 9
Number of Images: 1350
...
Running Iteration 0
Evaluating iteration 0, training accuracy 0.183099
...
Running Iteration 20
Evalutating iteration 20, training accuracy 0.726619
...
Running Iteration 40
Evalutating iteration 40, training accuracy 0.955556
...
Running Iteration 60
Evalutating iteration 60, training accuracy 0.993289
...
Running Iteration 80
Evalutating iteration 80, training accuracy 1
...
Running Iteration 100
Evalutating iteration 100, training accuracy 1
...
Running Iteration 119
Final Test Accuracy: 1
```

Fig. 8. Tensorflow log

Included here are some Tensorboard graphs showing the improvement of the model as it was being generated, as well as an overall tensor graph of the system:

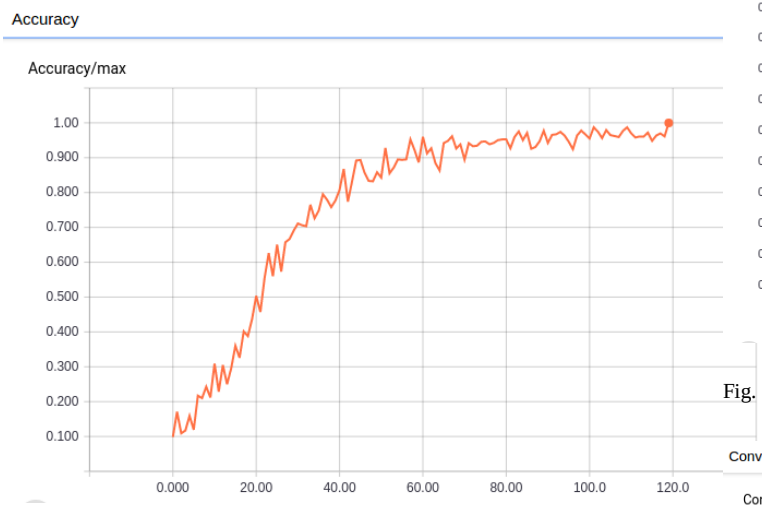

Fig. 9. Model Accuracy over Time (part 1)

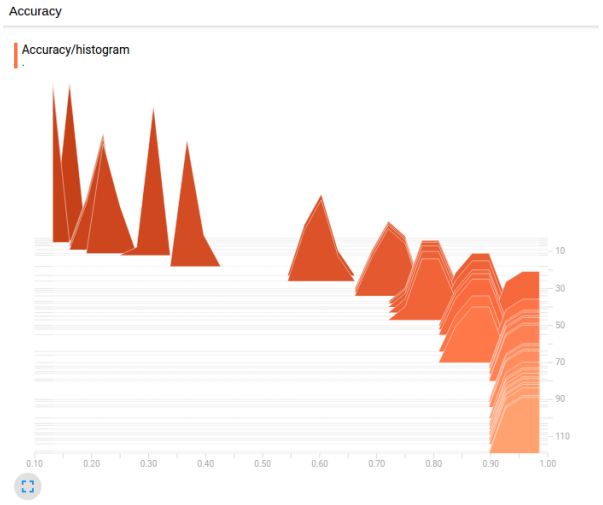

Fig. 10. Model Accuracy over Time (part 2)

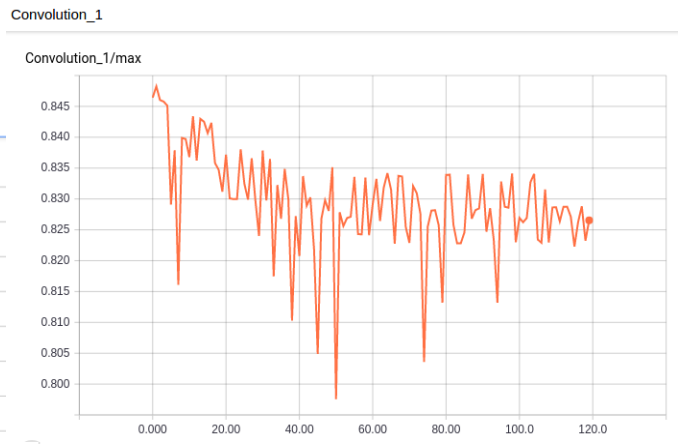

Fig. 11. First Convolutional Layer Results over Time

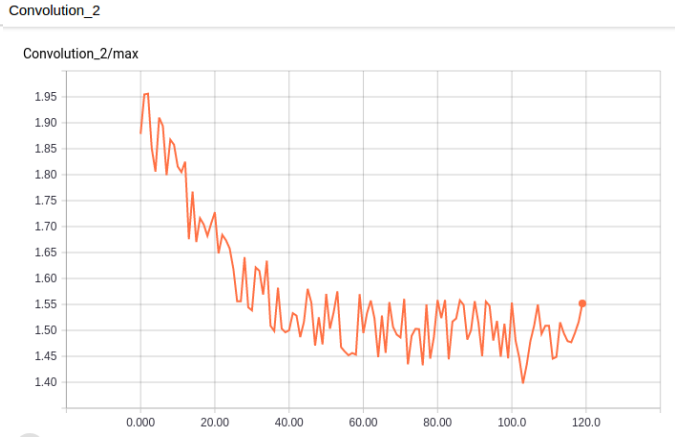

Fig. 12. Second Convolutional Layer Results over Time

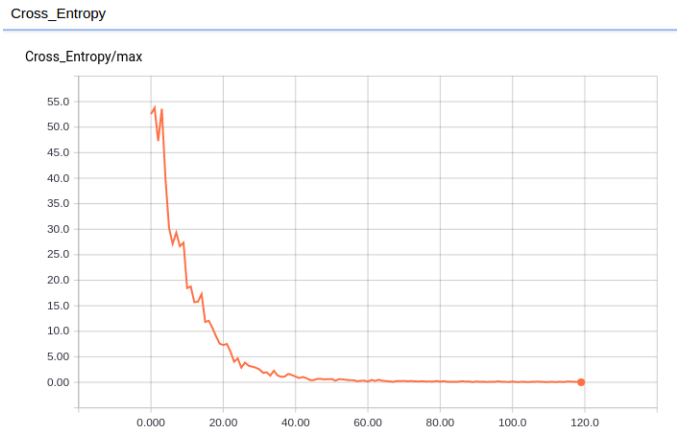

Fig. 13. Cross Entropy Function over Time

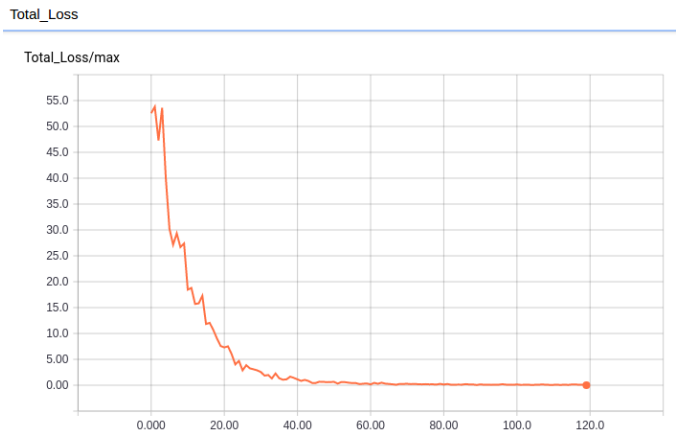

Fig. 14. Total Loss Function over Time

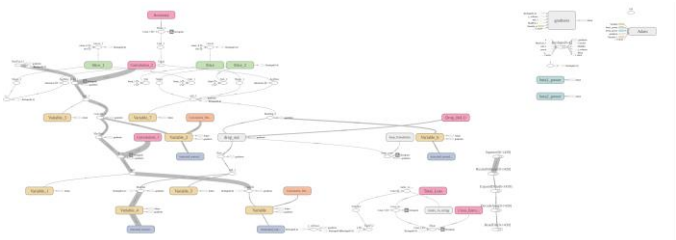

Fig. 15. Tensorflow Graph Diagram

G. *Performance Comparison*

Because Clarifai is an Open API to which we did not contribute, it does not bear discussion in this paper. It's accuracy for our data set is so poor as to not even be relevant for our use case.

We aim to compare several factors regarding performance: Time to Train Model and Time to Generate Image Prediction. Because of hardware limitations for the systems being used, both systems were bound in terms of how much data they could process, so it's worth reiterating that our dataset of 1350 training images and 90 test images was the hard limit of what could be processed without creating crashes in Spark. What is interesting is that each approach suffered from its own form of limitations: Spark could not handle a sheer volume of images, though it did not have a problem with the image size, since the result of each image was a feature histogram. Tensorflow did not have a problem with image volume, but rather struggled with image size, since larger images resulted in larger node counts required for the training model.

In terms of training time, the Spark model trains in about an average of 20 minutes. The longest portion of time in that process is the initial training data feature extraction and the Random Forest Generation time. Tensorflow, in contrast, took 120 iterations to ensure we could get the result desired, and this approach takes approx. 120 minutes while maximizing CPU usage for the Virtual Machine running the process. So, the Spark model clearly outperforms the Tensorflow model in terms of training time and training hardware requirements.

In terms of model loading, and important element of the REST service capabilities, the Spark model loads typically in less than 2 seconds of time. Tensorflow takes approximately 35 seconds to load. This is a significant different in loading time, but it's worth noting that the resultant model for Tensorflow is significantly larger in file size (by a factor of nearly 100). There is virtually no different in the time taken to generate the prediction once the model is loaded.

As discussed previously, the Tensorflow model was more accurate, generating a full 100% accuracy. The Spark model is relying on many details in the underlying image to generate a high accuracy prediction. In environments with less visual contract, we believe it safe to predict that Spark would perform significantly worse in terms of accuracy than any sufficiently trained Tensorflow model.

VI. Discussion & Limitation

There were several interesting findings from our project, and several limitations we want to call out. One obvious benefit to our approach that we discovered was the ability to narrow the focus of our training data to a finite data set. Instead of needing to recognize all statues as statues, we designed a system that for a specific room can tell a user that they are looking a specific statue, Michelangelo's David for instance, among a room of statues.

One advantage this brings is that multiple datasets need not be comprehensive. A system can be modularized to invoke a specific model or graph based on the problem domain, rather than need to know all things for all images. This is advantageous, because it will allow such a system to be easily enhanced and expanded upon in the future with little overhead. Within several hours, a brand new training data set could be created, modeled, and ready for consumption without interrupting any other existing services.

There were still many limitations we faced with our project. The first stems from the system requirements. The reality is that to implement our project, machine learning is actually not the most efficient way to solve this problem. This problem could be solved much more simply using standard object collision detection. That is, a system that uses the user's current point of view reference then checks collisions with known artifacts in the space would be more efficient, if slightly harder to implement. It would also be able to give near instantaneous feedback to the user, rather than needing to rely on external services.

That said, this project was about finding unique adaptations of image recognition techniques, which it does do. I think it is just worth noting that for this particular problem, I believe image recognition is not an ideal choice, which was not a known outcome when we began to prepare this project.

In continuing to discuss system requirements limitations, the requirement of using the Google Conversation API was a poor choice. While the API is interesting, it is poorly suited to this project, and I believe poorly suited to the type of solution we want to implement. In theory, you have a classroom full of students, all trying to use voice commands to interact with their environment. The risk of collisions and accidental invocation

of the voice system feels high, even without assuming any deliberate attempt by other students to interrupt the current user's experience.

Secondarily, the Google Conversation API makes a terrible middleman. While it is very simple to connect the Conversation with webhooks to process tasks more intelligently, the goal of this project was to invoke any one of three tasks. For any task that takes more than a matter of seconds, this is a poor choice, as the voice interaction service will respond that the transaction did not respond well before an actual response is received. Additionally, the API is a poor choice to serve as a middleware service. Our project needs to send a screenshot to one or all of three services for processing, so for the Conversation API to be even moderately relevant, the image content must be passed as part of that request, and must then be transferred from that request to other REST requests, which was very difficult to achieve, and outside the standard scope of usage for this API. In the author's opinion, the requirement to use this API was not only not required for this project, but actually counterproductive to the project.

We finally wish to discuss the technology limitations we encountered during this project. It should be stated that for many of the technologies presented during the semester, the authors had little to no exposure to those technologies and languages prior to beginning this project. In addition, much of the tutorial code design to ease adoption of these technologies was flawed or limited in application. Much of the provided Tensorflow code was completely unadaptable, and our code had to be rewritten almost from the ground up. There were also significant instabilities in the libraries used. Our Spark model would routinely crash if provided more than 800 training images, regardless of image size. The Tensorflow model was very resource intensive to generate, and we had to scale our image size to allow the model to be generated without overloading our limited system resources. Both of these limitations could be overcome with more robust hardware configurations and a more stable model code base.

Another technology limitation relates to implementing access to these models through a RESTful web-service. Tensorflow was unable to allow us to pre-load the model used for image analysis, which means that every request to Tensorflow was required to load the model from scratch, which takes around 30 seconds each time. This is an unacceptable level of performance. It was not clear whether Python, Flask, or Tensorflow were the source of these limitations, but significant effort was expended trying to pre-load the Tensorflow Graph on server initiation, but not successful approach was found that could retain access to the model inside a REST host.

VII. Conclusion

Broadly speaking, there are multiple conclusions that can be drawn from our research and development of this project:

- The Deep Learning approach has the capability to significantly outperform a standard Machine Learning approach.
- Both custom Deep Learning and Machine Learning approaches significantly outperform and open API not specific tailored to our data set.
- Generating a meaningful VR environment and training data set can be made simpler by confining the known universe of objects to be recognized to the content of a single VR panoramic image.
- While the Google Conversation API is powerful, it is not well suited to certain applications.
- Standard Object Collision methods commonly used in 3D gaming models provide a simpler approach to interactive VR experiences than image recognition.

As discussed previously, we were able to achieve 94.4% accuracy with Spark MLib and 100% accuracy with Tensorflow for our test environment. we believe the Spark MLib results could be further improved with Higher Resolution imagery (using a more powerful 360° camera or image capture device), and with more stable code dependencies. Because of the specificity required in our data set, the Clarifai API was essentially unable to recognize any images meaningfully from our data set. I do not believe this would have improved with larger access to a paid version of the API based off a much more comprehensive image library. The reality is that any broad system designed to recognize large numbers of classifiers can be expected to outperform a small but highly specific classifier model.

We believe that the Virtual Field Trip problem is a meaningful one in the education domain. We also believe that enhanced the simple static VR environment to provide an interactive experience provides great value to that experience. While we are not entirely convinced that Deep Learning or Machine Learning approaches as ideally suited to this project, we were able to use them relatively effectively to achieve the desired effect.

Acknowledgment

This work was done in partial fulfillment of the requirements of CS5542: Big Data Analytics and Apps, CSEE Department, University of Missouri – Kansas City (Spring 2017). Instructors: Dr. Yugyung Lee, Chandra Shekar, Mayanka, TAs: Maddula, Manikanta, Peddinti, Sudhakar Reddy, Vadlamudi, Naga K.

References

[1] Castrounis, Alex. Machine Learning: An In-Depth Guide - Overview, Goals, Learning Types, and Algorithms. Jan 27, 2016. http://www.innoarchitech.com/machine-learning-an-in-depth-non-technical-guide/?utm_source=kdnuggets&utm_medium=post&utm_content=originallink&utm_campaign=guest

[2] Clarifai – Image & Video Recognition API. https://clarifai.com/

[3] MLib | Apache Spark. http://spark.apache.org/mllib/

[4] Tensorflow. https://www.tensorflow.org/

[5] Oculus Rift | Oculus. https://www.oculus.com/rift/

[6] Google Cardboard | Google VR. https://vr.google.com/cardboard/

[7] Samsung Gear VR with Controller. http://www.samsung.com/global/galaxy/gear-vr/

[8] Google Expeditions. https://edu.google.com/expeditions/

[9] Virtual Fieldtrips and Scavenger Hunts. http://www1.ccs.k12.in.us/clm/media-center/fieldtrips

[10] NYT VR – Virtual Reality. https://play.google.com/store/apps/details?id=com.im360nytvr&hl=en

[11] Titans of Space Cardboard VR. https://play.google.com/store/apps/details?id=com.drashvr.titansofspacecb&hl=en

[12] Virtual reality field trips offer black history experiences for KC students. http://www.startlandnews.com/2017/02/virtual-reality-field-trips-offer-black-history-experiences-kc-students/