CS 5542 Big Data Analytics and Applications

Kansas City Fountain Monument Identifier – Report #3

Team #2 AJNN: #4 Jacqulyn Batson | #20 Chia-Hui Amy Lin | #36 Nickson Sigei | #46 Nancy Yang

---

❖ Project Management (Plan + Project Timelines, Members, Task Responsibility)

Implementation status report

✓ Work Completed

- <u>Description</u>: Finished setting up all three models (Clarifai, Spark, Tensorflow) and hooked them up with Android App. Still trying to clear the mud on the Google Conversation API. Three models are all connected with the App. Some troubleshoots have been done on the functionality.

- <u>Responsibility</u> (Task – Person):
  - ➢ <u>Nancy:</u>
    Wrap up part of the report.
  - ➢ <u>Amy:</u>
    Load in all three fountain datasets to generate the Retrain Inception Model in Tensorflow. Write, compile and format the final report.
  - ➢ <u>Jackie:</u>
    Added Tensorflow functionality to Android app. Fix possible errors and troubleshooting.
  - ➢ <u>Nickson:</u>

- <u>Time taken</u> (hours): approximately 60+ hours since the report 2.
- <u>Contributions</u>(members %): Jackie 31% | Nancy 28% | Amy 31% | Nick 10%

✓ Work Stalled

- Google Conversation part.

✓ Issues/Concerns

Google conversation part is still deep in the mud. Have the general idea of how to get it up with our app but things aren't quite set up for solely the conversation part. Nick somehow has been drilling on this task but nothing is being updated ever since Spring Break. This will definitely be our major issues to overcome within the next submission or more like to wrap up the project.

3 Deep Learning Approaches for Identification of 3 Kansas City Fountains

Amy Lin, Jackie Batson, Nancy Yang, Nickson Sigei

Abstract

Three models are built to develop image recognition programs – 1. Clarifai: an external API to automatically analyze image data, 2. Spark: train and test decision tree and random forest models in machine learning, 3. Tensorflow: deep learning method by using pre-trained model to do further training on multilayer neural networks.
Three sets of fountain images in Kansas City are used as the dataset, including Children's Fountain, J.C. Nichols Memorial Fountain and The Muse of Missouri. An Android APP is developed to enable user to identify one of the three fountains from our dataset by using their own device. Also, three options are provided to implement a model based on user's own choice. The information of the specific fountain will be returned and displayed through the app.

1. Introduction

The first fountain in Kansas City was built to serve dogs, horses and birds in the late 1800s. Ever since, drinking fountains began to erected throughout the city. This in turn represent what earned Kansas City the well-known moniker - the City of Fountains. The project is built in hope to assist tourists or local people to learn more about the fountains existing around this beautiful city. Furthermore, it will be the middleman of telling the background stories in just one click and enable people to get on the ride of the artistic sprit in Kansas City within fingers. By taking a picture of a fountain, relative information(author, name of the fountain, inspiration of creating this art, background story) about this piece of art will return to the user.

2. Related work

Google Arts & Culture (https://www.google.com/culturalinstitute/beta/u/0/). The app collaborates with over 1200 international museums, galleries and institutions from 70 countries to make their exhibits available for anyone. The ultimate goal of our project is to have the same amount of data covered similar to this app. However, it is only limited to visualization of a certain piece of art on an app. It lacks the function that can take in an input and output the information and details. This app is a huge art & culture dataset designed to fit the artistic interest whenever or wherever one wants to enjoy a piece of art. From the aspect of image recognition and deep learning, our project can come in handy.

3. Proposed work

We only focus on training and testing three fountains in Kansas City(J.C. Nichols Memorial Fountain, Muse of Missouri, and Children's Fountain) for efficiency purpose.
Three models are build separately through different approaches. All models are hooked up with the Android APP. The "take picture" button is for taking a photo as an input. Three models are assigned to three buttons with their own name for output options (Clarifai, Spark, TensorFlow). The button that is picked by the user will trigger the model to analyze the input from the user and return the predict result accordingly.

a. Proposed models

- Clarifai
  The Clarifai API serves as an image recognition server between two entities. It takes images or any input from one end to the service and

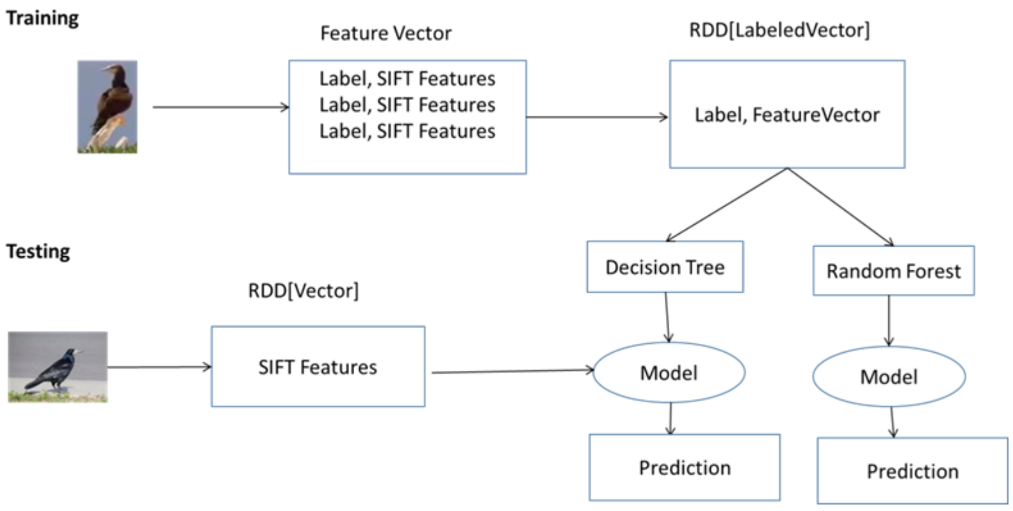


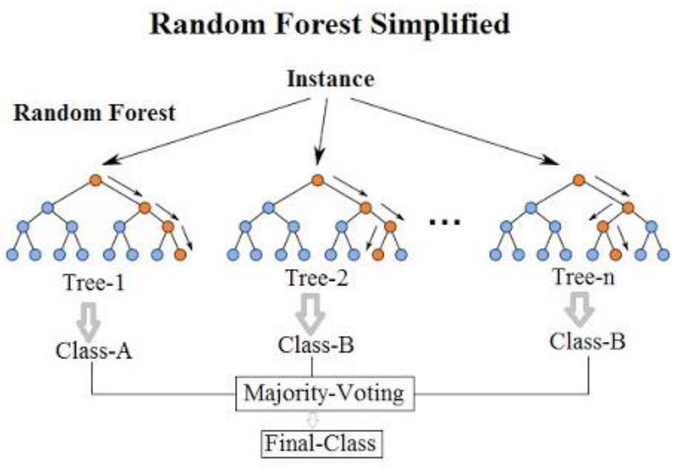
Image 1: Training and testing process of SIFT Feature Extractions

returns a prediction. Prediction obtains by the user will depend on the model that will be able to handle the input.

- SPARK

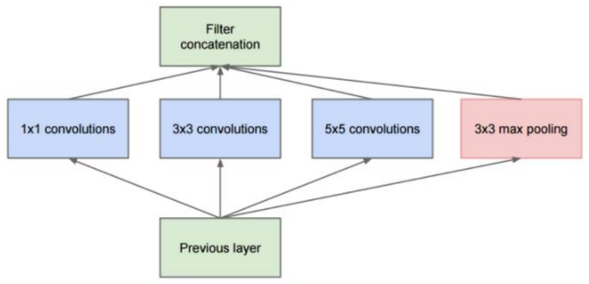
  Feature extraction and image classification model are implemented in SPARK.
  A series of SIFT feature vector is extracted through the feature extraction. The process is shown in Image 1 above.

  Random Forest algorithm is used for classification images. Illustration of Random Forest algorithm can be seen in the following diagram:

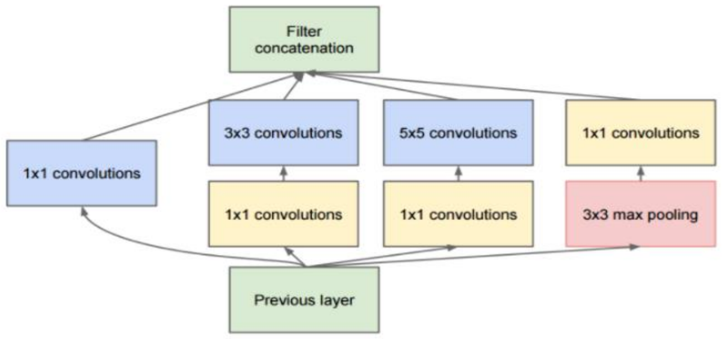


- TensorFlow

  GoogleNet Inception model is built through Tensorflow.



(a) Inception module, naïve version



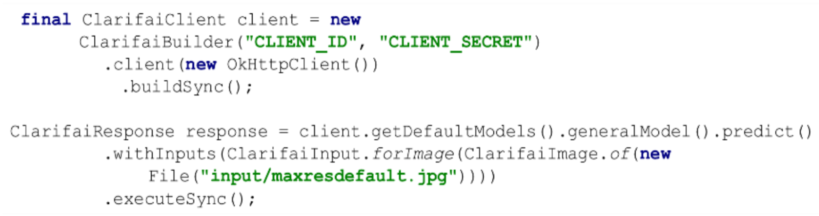
(b) Inception module with dimension reductions

In the beginning, ImageNet was not trained on any input data. Script will load the pre-trained Inception v3 model, remove the old final layer, and trains a new layer based on our own datasets. Training the final layer will give us the bottleneck data. A 'Bottleneck,' is an informal term we often use for the layer that is before the final output layer that performs the classification. Every image is reused multiple times during training. Time taken to calculate the layers behind the bottleneck for each image is quite significant. Therefore, by caching the outputs of the lower layers on disk, the layer won't have to be repeatedly recalculated.

b. Algorithms and Pseudocode

- <u>Clarifai</u>

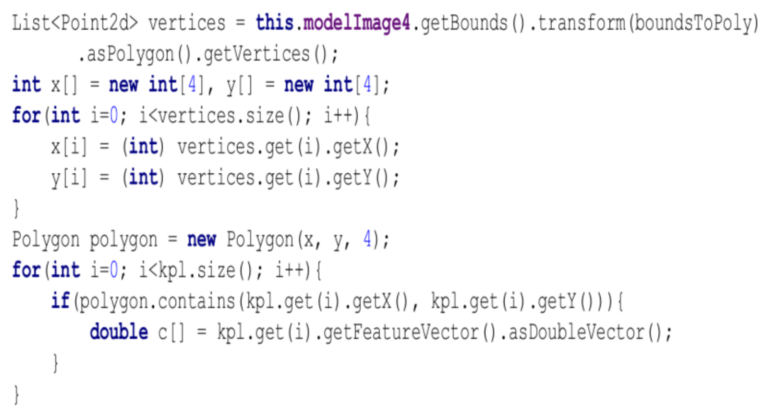
  To connect with the Clarifai API, the user have to create a client ID and secret token from the client side. After applying the two key info, the server will be hooked up with the program to run the model. Pseudocode is shown in the following image:



```
 final ClarifaiClient client = new
      ClarifaiBuilder("CLIENT_ID", "CLIENT_SECRET")
         .client(new OkHttpClient())
           .buildSync();

ClarifaiResponse response = client.getDefaultModels().generalModel().predict()
           .withInputs(ClarifaiInput.forImage(ClarifaiImage.of(new
                File("input/maxresdefault.jpg"))))
           .executeSync();
```

- <u>Feature Extraction</u>

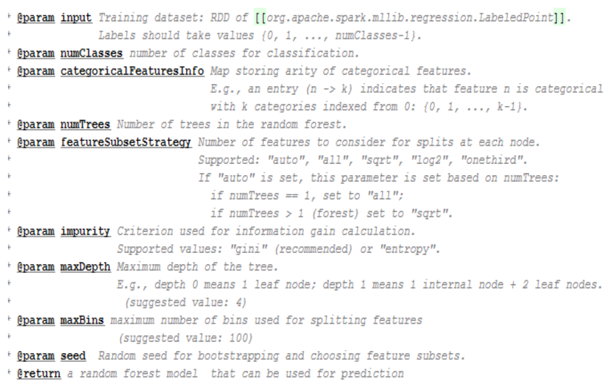
  Extract Features inside the polygon drawn from the picture.



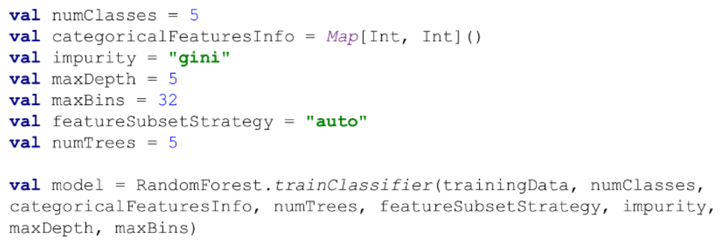
```
List<Point2d> vertices = this.modelImage4.getBounds().transform(boundsToPoly)
        .asPolygon().getVertices();
int x[] = new int[4], y[] = new int[4];
for(int i=0; i<vertices.size(); i++){
    x[i] = (int) vertices.get(i).getX();
    y[i] = (int) vertices.get(i).getY();
}
Polygon polygon = new Polygon(x, y, 4);
for(int i=0; i<kpl.size(); i++){
    if(polygon.contains(kpl.get(i).getX(), kpl.get(i).getY())){
        double c[] = kpl.get(i).getFeatureVector().asDoubleVector();
    }
}
```

- <u>Random Forest</u>

  Algorithms and pseudocode of Random Forest can be seen below:

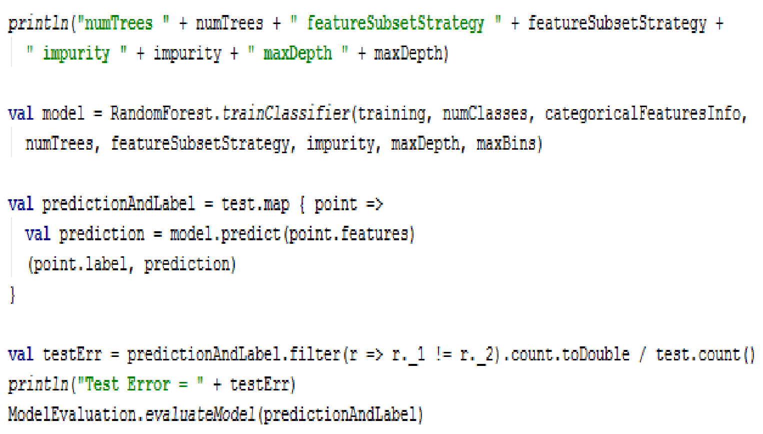


```
* @param input Training dataset: RDD of [[org.apache.spark.mllib.regression.LabeledPoint]].
*              Labels should take values {0, 1, ..., numClasses-1}.
* @param numClasses number of classes for classification.
* @param categoricalFeaturesInfo Map storing arity of categorical features.
*                                E.g., an entry (n -> k) indicates that feature n is categorical
*                                with k categories indexed from 0: {0, 1, ..., k-1}.
* @param numTrees Number of trees in the random forest.
* @param featureSubsetStrategy Number of features to consider for splits at each node.
*                              Supported: "auto", "all", "sqrt", "log2", "onethird".
*                              If "auto" is set, this parameter is set based on numTrees:
*                                if numTrees == 1, set to "all";
*                                if numTrees > 1 (forest) set to "sqrt".
* @param impurity Criterion used for information gain calculation.
*                 Supported values: "gini" (recommended) or "entropy".
* @param maxDepth Maximum depth of the tree.
*                 E.g., depth 0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.
*                  (suggested value: 4)
* @param maxBins maximum number of bins used for splitting features
*                 (suggested value: 100)
* @param seed  Random seed for bootstrapping and choosing feature subsets.
* @return a random forest model  that can be used for prediction
```



```
val numClasses = 5
val categoricalFeaturesInfo = Map[Int, Int]()
val impurity = "gini"
val maxDepth = 5
val maxBins = 32
val featureSubsetStrategy = "auto"
val numTrees = 5

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

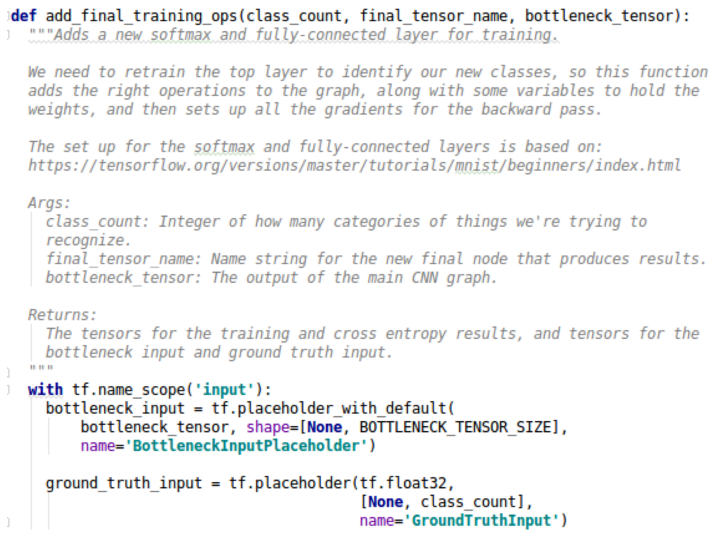
Parameters of setting of the forest are determined by the total number of image classes. Training the Classifier Based on Random Forest:



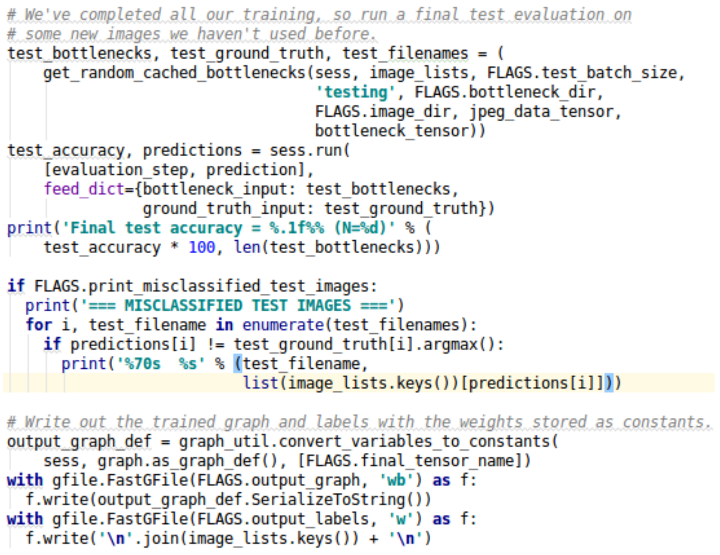
```
println("numTrees " + numTrees + " featureSubsetStrategy " + featureSubsetStrategy +
  " impurity " + impurity + " maxDepth " + maxDepth)

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)
```

- <u>TensorFlow</u>

Set up function for final training ops.



```
def add_final_training_ops(class_count, final_tensor_name, bottleneck_tensor):
  """Adds a new softmax and fully-connected layer for training.

  We need to retrain the top layer to identify our new classes, so this function
  adds the right operations to the graph, along with some variables to hold the
  weights, and then sets up all the gradients for the backward pass.

  The set up for the softmax and fully-connected layers is based on:
  https://tensorflow.org/versions/master/tutorials/mnist/beginners/index.html

  Args:
    class_count: Integer of how many categories of things we're trying to
    recognize.
    final_tensor_name: Name string for the new final node that produces results.
    bottleneck_tensor: The output of the main CNN graph.

  Returns:
    The tensors for the training and cross entropy results, and tensors for the
    bottleneck input and ground truth input.
  """
  with tf.name_scope('input'):
    bottleneck_input = tf.placeholder_with_default(
        bottleneck_tensor, shape=[None, BOTTLENECK_TENSOR_SIZE],
        name='BottleneckInputPlaceholder')

    ground_truth_input = tf.placeholder(tf.float32,
                                        [None, class_count],
                                        name='GroundTruthInput')
```



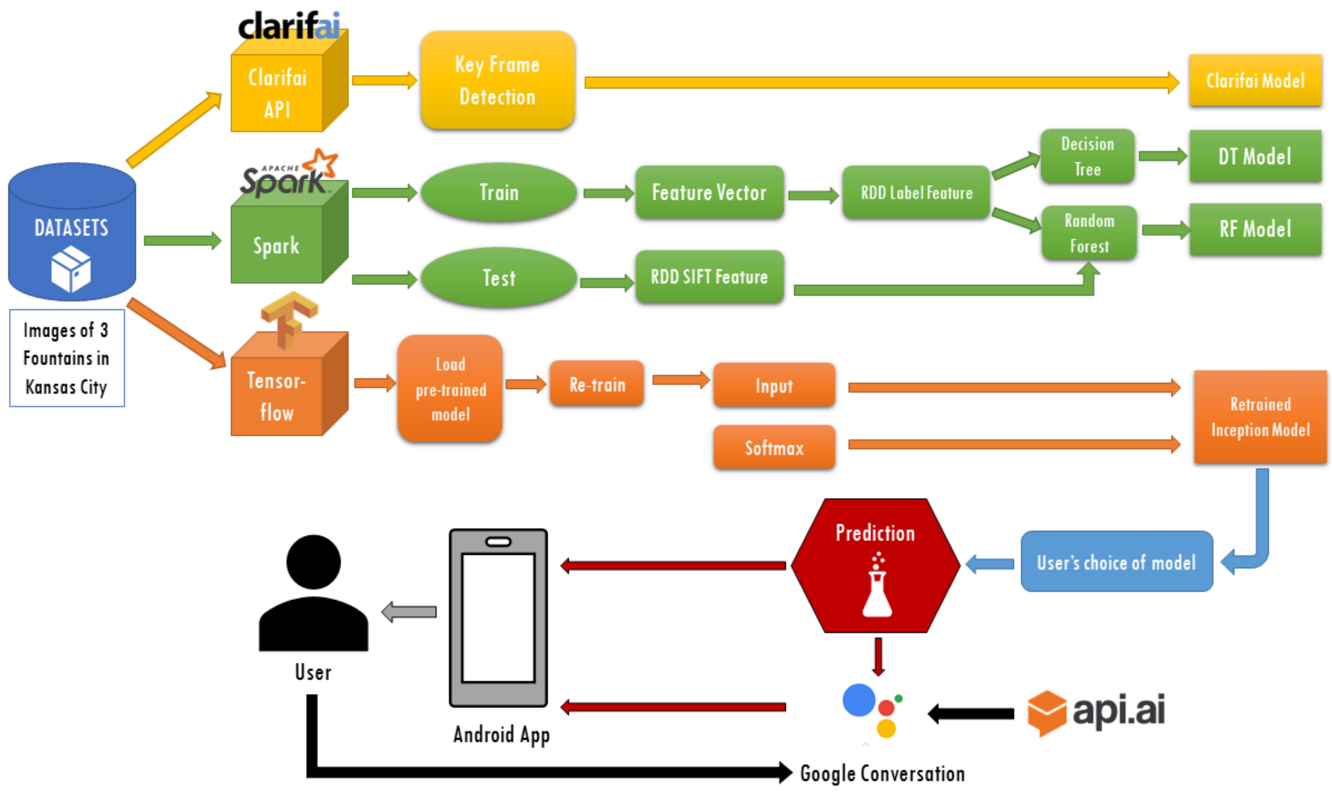
```
# We've completed all our training, so run a final test evaluation on
# some new images we haven't used before.
test_bottlenecks, test_ground_truth, test_filenames = (
    get_random_cached_bottlenecks(sess, image_lists, FLAGS.test_batch_size,
                                  'testing', FLAGS.bottleneck_dir,
                                  FLAGS.image_dir, jpeg_data_tensor,
                                  bottleneck_tensor))
test_accuracy, predictions = sess.run(
    [evaluation_step, prediction],
    feed_dict={bottleneck_input: test_bottlenecks,
               ground_truth_input: test_ground_truth})
print('Final test accuracy = %.1f%% (N=%d)' % (
    test_accuracy * 100, len(test_bottlenecks)))

if FLAGS.print_misclassified_test_images:
  print('=== MISCLASSIFIED TEST IMAGES ===')
  for i, test_filename in enumerate(test_filenames):
    if predictions[i] != test_ground_truth[i].argmax():
      print('%70s  %s' % (test_filename,
                          list(image_lists.keys())[predictions[i]]))

# Write out the trained graph and labels with the weights stored as constants.
output_graph_def = graph_util.convert_variables_to_constants(
    sess, graph.as_graph_def(), [FLAGS.final_tensor_name])
with gfile.FastGFile(FLAGS.output_graph, 'wb') as f:
  f.write(output_graph_def.SerializeToString())
with gfile.FastGFile(FLAGS.output_labels, 'w') as f:
  f.write('\n'.join(image_lists.keys()) + '\n')
```

----

4. Implementation and Evaluation

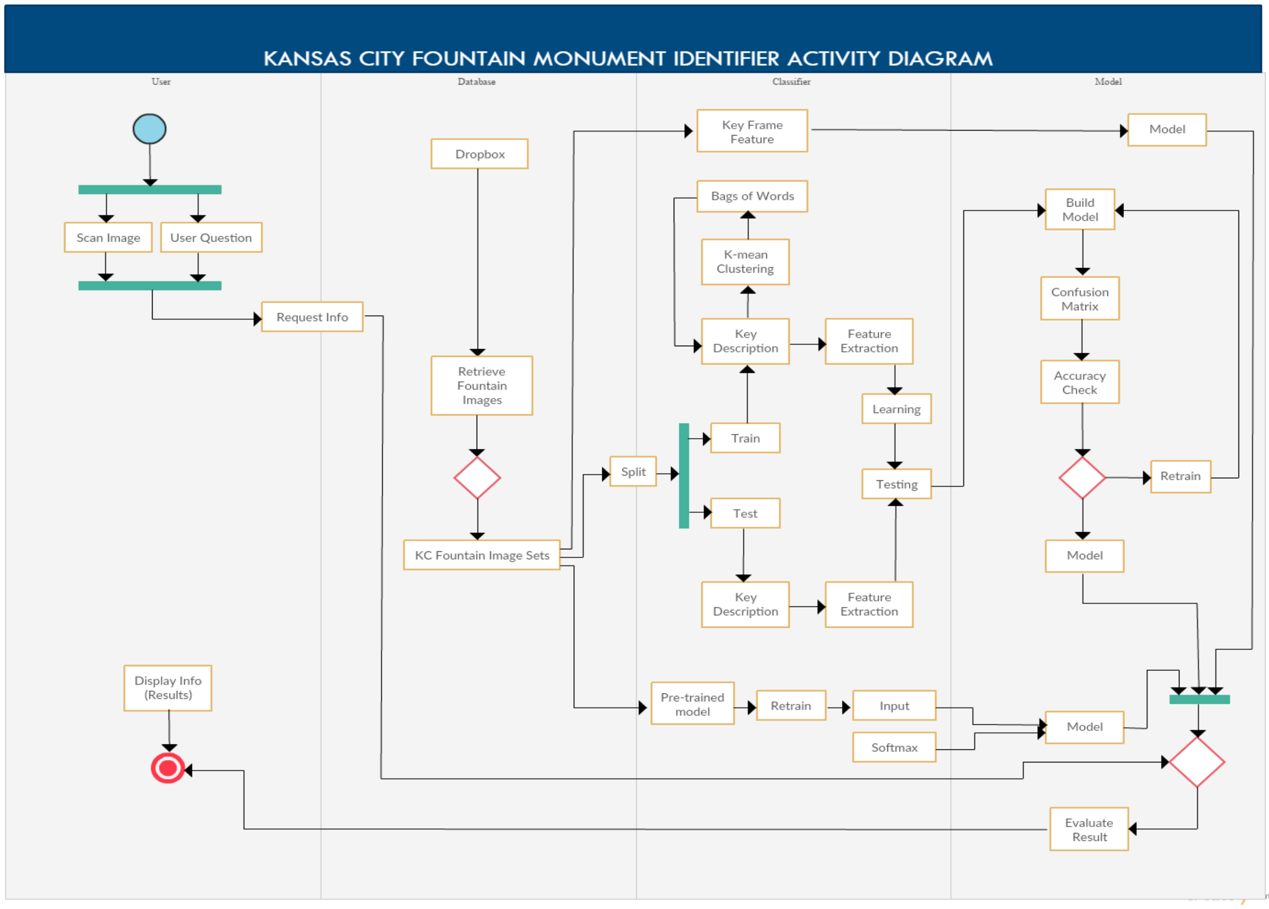
a. System Design and Implementation

a-i. Software architecture:

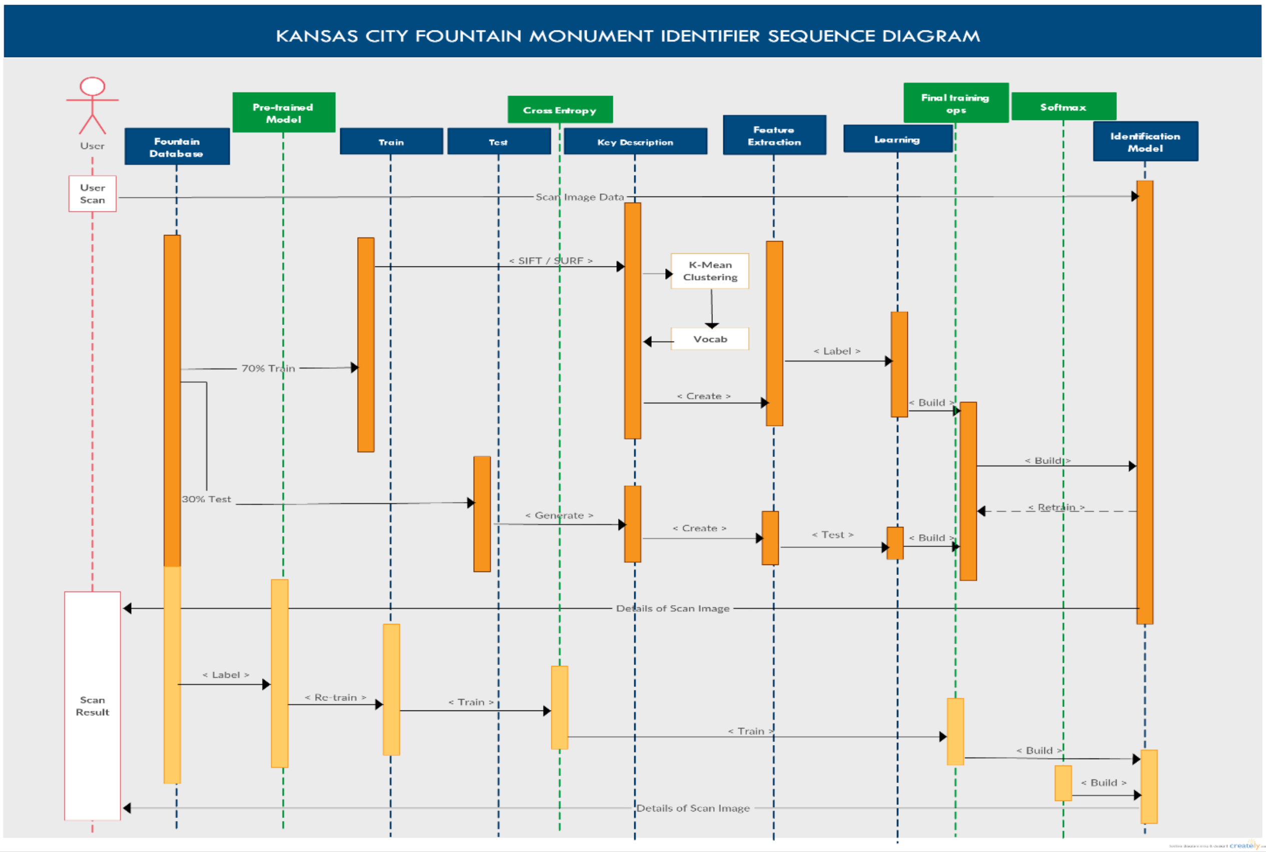


a-ii. UML Diagrams:

Activity Diagram



Sequence Diagram



a-iii. Implementation details

Download app through an Android device. Image(input) can be taken by the camera on user's device. Input will be parse, extract, or break down into training/testing, pixel features, or go directly into a pre-trained modeling for further training. This will be determined by the user's choice of which model that he/she wants to perform. A fair prediction will be given back to the user and displayed on the UI of the app.

b. *Evaluation and Results:*

b-i. Evaluation plan:

- Datasets
  Images of three fountains (Children's Fountain, J.C. Nichols Memorial Fountain, The Muse of Missouri) are included. Maximum number of image category is 63 and the rest of the them are approximately 40. Confusion matrix is used to show how accurate each category can be identified.

- System Speculation
  Train time for each model is no more than 30 minute. We use a i7-6700HQ processor and 16 RAM CPU to build model using three main approaches.

- Measurement
  Accuracy of recognizing the correct image is used to evaluate which model is a better fit than the other.

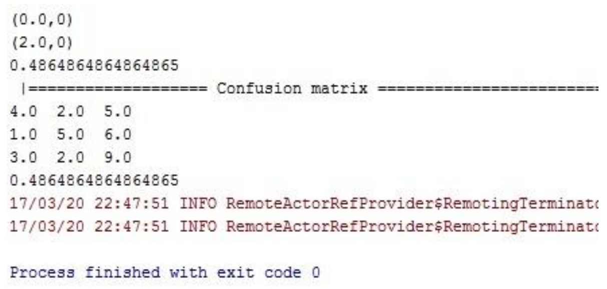
b-ii. Evaluation results(Comparative evaluation):

Comparative Evaluation for three different approaches Performance (accuracy runtime, space requirements, scalability…)

- <u>Clarifai</u>

Run time through this approach takes a lot longer than the other two approaches. The model will scan through an image several times to retrieve possible contents. Downside of this approach is lack of sufficient datasets might get a low accuracy on retrieving the accurate keywords containing in a certain image.

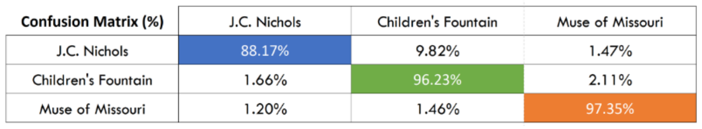
- <u>Spark</u>
  For Spark implementation, we have a fair accuracy of 46.84%. A confusion matrix is generated as well to evaluate the percentage of correctly classified images.



```
(0.0,0)
(2.0,0)
0.4864864864864865
 |==================== Confusion matrix ==========================
4.0  2.0  5.0
1.0  5.0  6.0
3.0  2.0  9.0
0.4864864864864865
17/03/20 22:47:51 INFO RemoteActorRefProvider$RemotingTerminat
17/03/20 22:47:51 INFO RemoteActorRefProvider$RemotingTerminat

Process finished with exit code 0
```

- <u>Tensorflow</u> <u>(Retrained Inception Model)</u>

The accuracy for the tensorflow model is 92.3% with only 200 steps of training. Images correctly place in its category have at least 88% accuracy. For a small or limited of datasets we have, this model will be the perfect fit for a decent and accurate prediction!



Confusion Matrix (%)	J.C. Nichols	Children's Fountain	Muse of Missouri
J.C. Nichols	88.17%	9.82%	1.47%
Children's Fountain	1.66%	96.23%	2.11%
Muse of Missouri	1.20%	1.46%	97.35%

5. Discussion and Limitation

The three fountain data we collected have limited number of images for the same fountain (different angles, time taken, from different authors … etc). Furthermore, only 3 out of 200 registered fountains in Kansas City is included. Even though more data has the potential of overfitting the mode, it can improve the model accuracy with a just-about-right amount of images. For the Inception Model, it is possible to add more layers if we can run the deep learning approach on a GPU. However, for a CPU implementation, we still have a decent accuracy from the Inception Model.
Ultimately, we would like to include all fountains in Kansas City as our dataset.

6. Conclusions

Deep learning and wide range usage of API have been trending these days. It's amazing how we can use our own laptop with a powerful enough CPU to run a machine learning task. Open source APIs enable us to extend functionality by simply hooking up the server with customized key and token. For image identification/classification, techniques mentioned ahead are combined or cross-used to get the most accurate result. A decent or well-trained model can thus allow us for further usage on future tasks.

For our application, we should get good results from the Tensorflow model. Wherein, Clarifai and Spark will need more datasets to improve the accuracy. In the future, we would like to complete all 200 registered fountains in Kansas City and expand more to nationwide fountains. Also, app-wise we would like to provide more functionality such as connecting with Google Cardboard for Virtual Reality environment so user can browse around like they're actually there looking at the fountain.

REFERENCE

https://www.google.com/culturalinstitute/beta/u/0/
https://www.visitkc.com/visitors/things-do/attractions/kansas-city-city-fountains#sm.001krhznt1447dphr0r2iproiqv8s
https://www.youtube.com/watch?v=VxhSouuSZDY
https://sites.google.com/site/yorkyuhuang/home/research/machine-learning-information-retrieval/scene-classification
http://www.mathworks.com/help/vision/ug/image-classification-with-bag-of-visual-words.html
http://www.ijircce.com/upload/2013/april/21_V1204057_A%20Comparison_H.pdf