3 Deep Learning Approaches for Identification of 3 Kansas City Fountains

Jackie Batson[1], Chia-Hui Amy Lin[2], Nancy Yang[3], Nickson Sigei[4]

School of Computing and Engineering, University of Missouri - Kansas City

Kansas City, USA

***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 Childrens 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 users own choice. The information of the specific fountain will be returned and displayed through the app.**

I. 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 spirit 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.

II. 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 and 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.

III. 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

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

- 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 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:

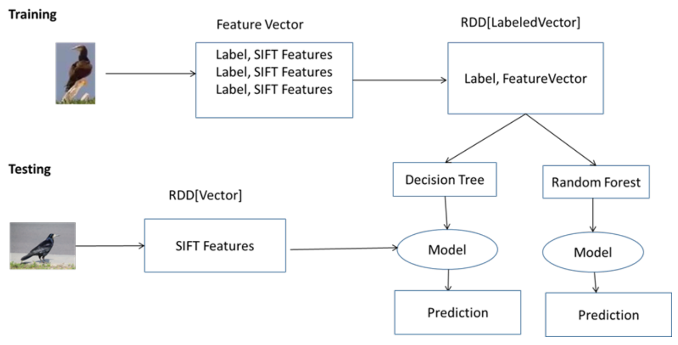


Fig. 1. Spark SIFT feature extraction training and testing process

Decision Tree algorithm is used for classification images. A decision tree is a flow-chart-like structure, where each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of a test and each leaf (or terminal) node holds a class label. The topmost node in a tree is the root node. There are many specific decision-tree algorithms. An example of Decision Tree algorithm can be seen in the following diagram:

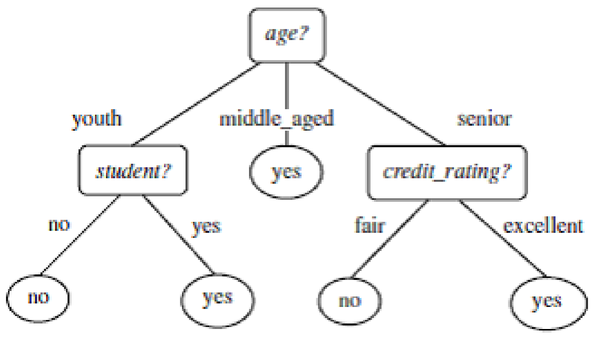


Fig. 2. Decision Tree algorithm

- Tensorflow:
  GoogleNet Inception model is built through Tensorflow.

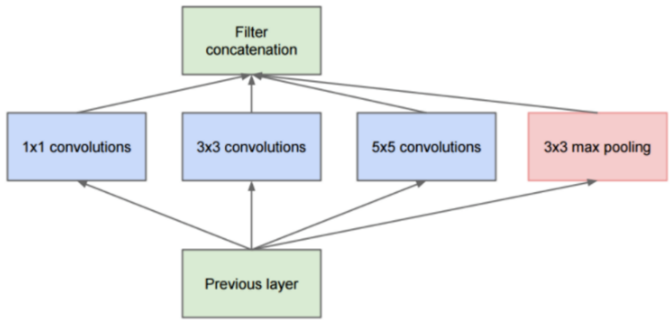


Fig. 3. Naive version of Inception Module

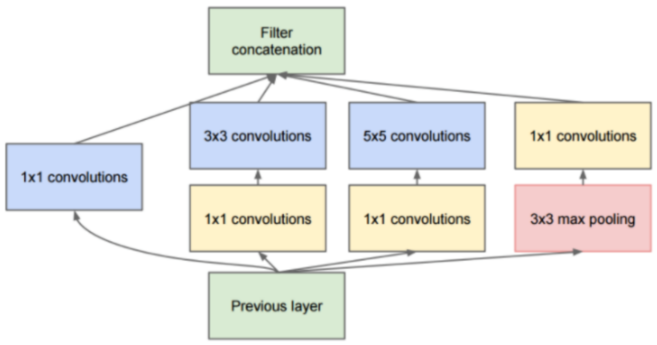


Fig. 4. Inception Module with dimension reductions

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 wont have to be repeatedly recalculated.

B. Algorithms and Pseudocode

- Clarifai:
  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("I27Ddv1UvE2gnbZa2hglGdbA930suypYPvWwwmGl",
"N9SPvnyizl4ZBX6PDgAqjR5kd_Nvnc-p0_BaTiGa")
     .client(new OkHttpClient()) // OPTIONAL. Allows customization of OkHttp by the
user
     .buildSync(); // or use .build() to get a Future<ClarifaiClient>
client.getToken();
```

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

Fig. 5. Pseudocode for Clarifai

- Feature Extraction:
  Extract Features inside the polygon drawn from the picture.

```
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();
            }
      }
```

Fig. 6. Pseudocode of polygon for extracting features

- Decision Tree:
  Algorithms and pseudocode of Decision Tree can be seen below:

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

val model = DecisionTree.trainClassifier(trainingData,numClasses,
categoricalFeaturesInfo, impurity, maxDepth, maxBins)
```

Parameters of setting up the forest are determined by the total number of image classes.

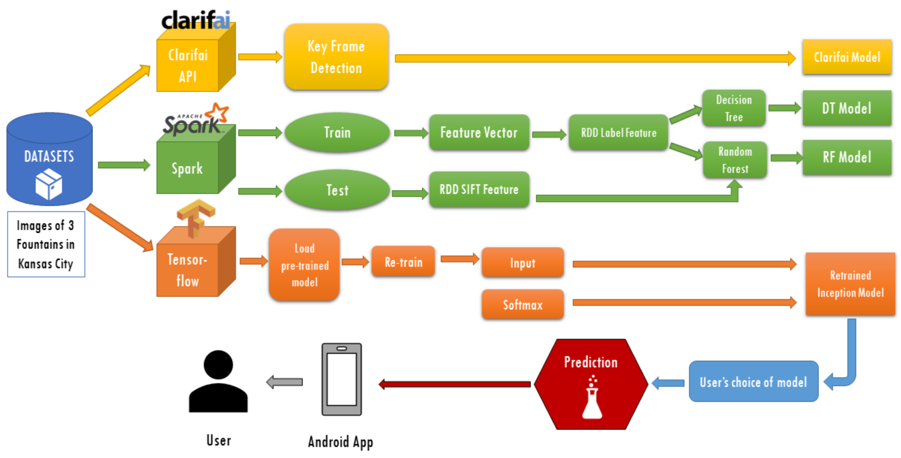
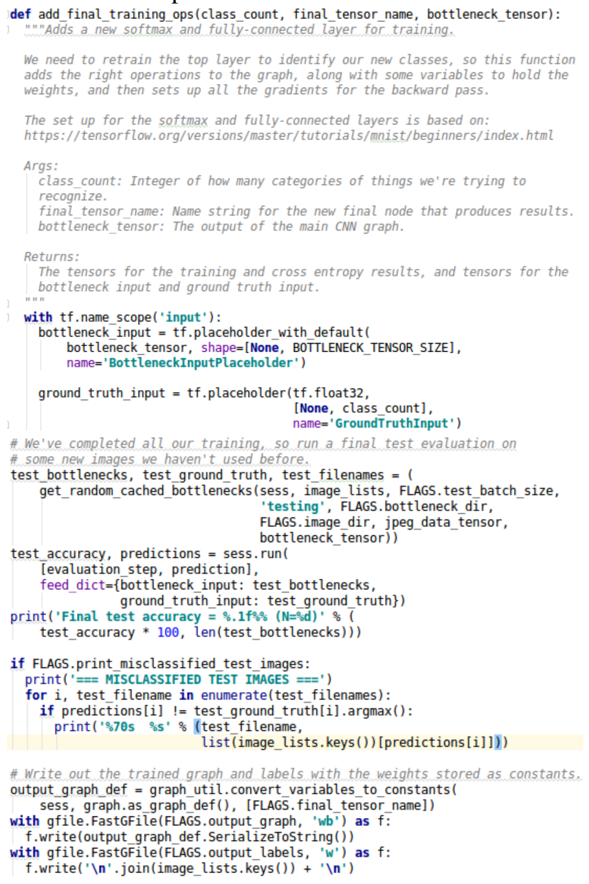


Fig. 7. Implementation workflow

- Tensorflow:
  Set up function for final training ops. (See Figure 8). Top layer has to be retrained so it will be able to identify new classes. The function shown below 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.



```
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')
```

Fig. 8. Pseudocode of final training ops in the Retrained Inception Model

IV. IMPLEMENTATION AND EVALUATION

A. System Design and Implementation

The overall workflow is in Figure 7 above. Three datasets are set up to train and test for three selected modules-Clarifai, Spark and Tensorflow. Clarifai digs out every possible key frames in an image. Spark is trained based on the datasets we contain. The result is tested by a small set of dataset to enable the model to learn. Decision tree algorithm is chosen as our final one for Spark. For Tensorflow, a pre-trained model is used and its last layer is cherry picked for further trained with the new inputs. The user will see the prediction of the model he/she prefers to perform.

B. Evaluation and Results

1) Evaluation plan:

- Datasets:
  Images of three fountains (Childrens 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.

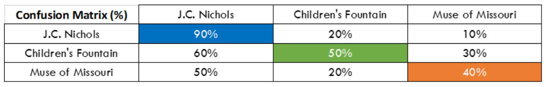
- Measurement:
  Accuracy of recognizing the correct image is used to evaluate which model is a better fit than the other.

2) Evaluation results:

- Clarifai:
  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.

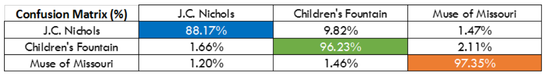
- Spark:
  For Spark implementation, we have a fair accuracy of 46.84%. A confusion matrix is generated as well to evaluate the percentage of correctly classified images.



Confusion Matrix (%)	J.C. Nichols	Children's Fountain	Muse of Missouri
J.C. Nichols	90%	20%	10%
Children's Fountain	60%	50%	30%
Muse of Missouri	50%	20%	40%

Fig. 9. Spark Confusion Matrix

- Tensorflow (Retrained Inception Model): The accuracy for the tensorflow model is 92.3% with 5000 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%

Fig. 10. Retrained Inception Model Confusion Matrix

V. 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. If the image taken is outside of the three categories we have, it will display or return the false information. Even though more data has the potential of overfitting the mode, it can certainly improve the Spark 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.

VI. CONCLUSIONS

Deep learning and wide range usage of API have been trending these days. Its 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 have a good result from the Tensorflow model as it is expected to be. 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 theyre actually there looking at the fountain.

REFERENCES

[1] Google Arts & Culture, https://www.google.com/culturalinstitute/beta/

[2] Kansas City is the City of Fountains, https://www.visitkc.com/visitors/things-do/attractions/kansas-city-city-fountains#sm.001krhznt1447dphr0r2iproiqv8s, VisitKC

[3] Inception Module, https://www.youtube.com/watch?v=VxhSouuSZDY, Udacity

[4] Yu Huang, BoW-based Image/Scene Classification with Naive Bayes Classifiers/SVMs,https://sites.google.com/site/yorkyuhuang/home/research/machine-learning-information-retrieval/scene-classification

[5] Steed, Anthony, et al. ”An In the WildExperiment on Presence and Embodiment using Consumer Virtual Reality Equipment.” IEEE transactions on visualization and computer graphics 22.4 (2016): 1406-1414.

[6] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. ”Deep learning.” Nature521.7553 (2015): 436-444.

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

[8] Szegedy, Christian, et al. ”Going deeper with convolutions.” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[9] Abadi, Martn, et al. ”Tensorflow: Large-scale machine learning on heterogeneous distributed systems.” arXiv preprint arXiv:1603.04467 (2016).

[10] Cordts, Marius, et al. ”The cityscapes dataset for semantic urban scene understanding.” arXiv preprint arXiv:1604.01685 (2016).

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed,Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. ”Going deeper with convolutions” (2015)