DATA 603 PROJECT REPORT

Image Classification and Reverse Image Search

GROUP 4

Abstract:

The primary goal we strived to achieve is the ability to process large amounts of data and convey the results of our analysis through informative visualizations. We used the UMBC Big Data Cluster to classify images of animals and birds using the MobileNet pre-trained model. We calculated accuracies for each label and plotted the distribution. Later, we performed Reverse Image Search and Image Ranking.

About the Dataset:

Google Open Image Dataset. This dataset consists of 9 million images divided into 15,387 classes. The dataset is split into a training set (9,011,219 images), a validation set (41,620 images), and a test set (125,436 images). Since we are using only a subset of this data, the size of the dataset is around 500 GB.

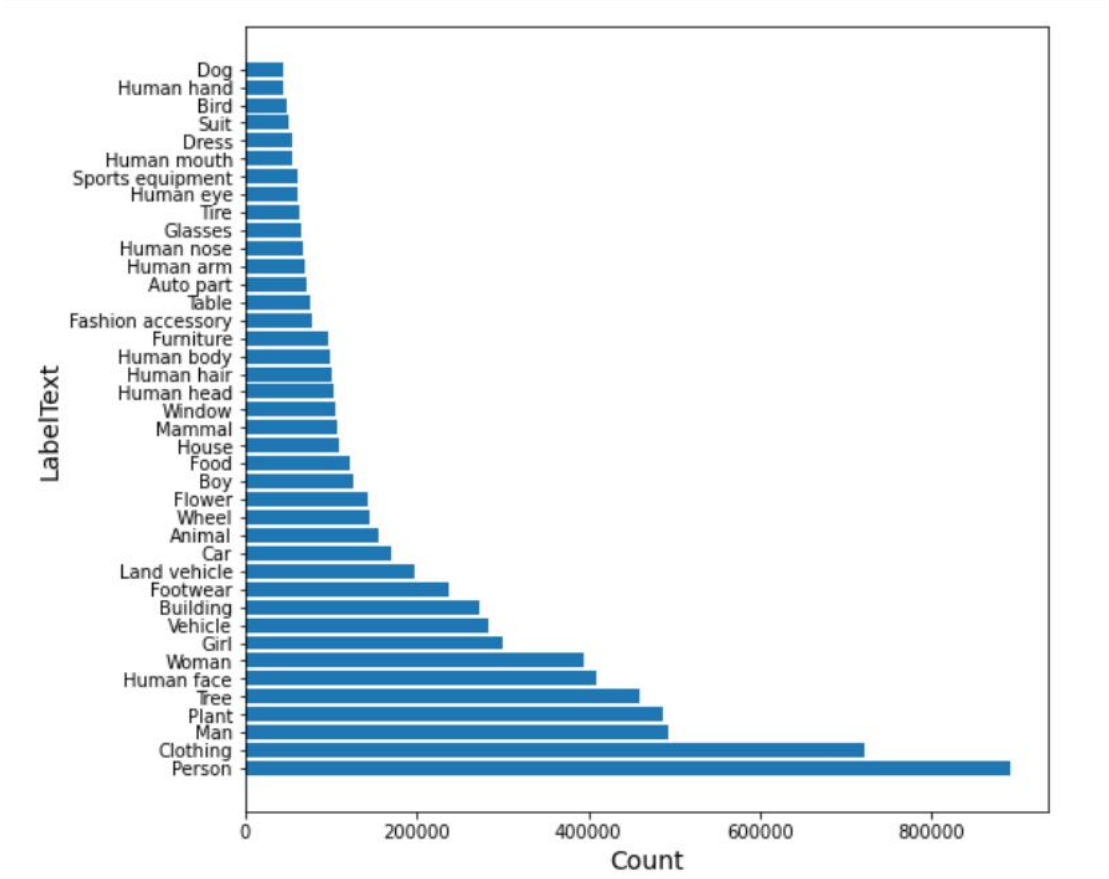
Exploratory Data Analysis (EDA):

Initially, before starting our analysis, we tried exploring the dataset to gain some meaningful insights into the data that could help us along the course of the project. We started by setting up HDFS, initializing Spark, setting our environment configuration, creating directories for weight files in our directory, installing packages like tensorflow, keras applications, spark.sql functions, numpy, pandas, matplotlib, os, etc. We used the class_descriptions_boxable.csv from metadata, image label data from labels.csv, the images.parquet file, and the bounding boxes data (test, train and validation) from the HDFS for our analysis.

We changed the column names for ease of use using the withColumnRenamed() function , checked for the consistency of data by extracting null values using .filter("column is null"), checked if every image had a label, found number of bounding boxes per image using the groupby function defined schema for labels by defining the data type of each column, and filtered the data for images having a confidence value greater than 0.99. We tried finding differences between the test, train, and validation subsets of the data. We found that both the test subset and validation subset had all their distinct images with a confidence greater than 0.99 whereas the train set had a lower ratio of images having such a high confidence. We found the number of occurrences of each label in the data frame using the groupby function. We plotted this distribution to understand the proportions of each label in the data and to shortlist our labels for classification based on these proportions.

We also tried finding the subsets/faucets present in the data frame for the 600 various labels which was organized into several categories and sub-categories. We found a JSON file (bbox_labels_600_hierarchy) which we explored to see how they are categorized.

A distribution of top 40 labels based on number of occurrences in the dataframe



Model - MobileNet:

Intro

MobileNets are based on a streamlined architecture that uses depthwise separable convolutions to build light weight deep neural networks. We introduce two simple global hyper-parameters that efficiently trade-off between latency and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem.

The general trend has been to make deeper and more complicated networks in order to achieve higher accuracy. However, these advances to improve accuracy are not necessarily making networks more efficient with respect to size and speed. In many real world applications such as

robotics, self-driving car and augmented reality, the recognition tasks need to be carried out in a timely fashion on a computationally limited platform. MobileNets primarily focus on optimizing for latency but also yield small networks.

MobileNets are built primarily from depthwise separable convolutions initially introduced in and subsequently used in Inception models to reduce the computation in the first few layers. Flattened networks build a network out of fully factorized convolutions and showed the potential of extremely factorized networks. A different approach for obtaining small networks is shrinking, factorizing or compressing pre-trained networks.

Architecture:

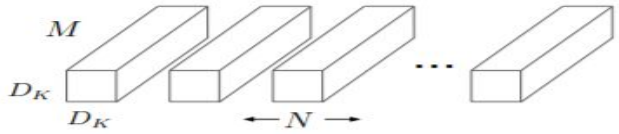
The MobileNet model is based on depthwise separable convolutions which is a form of factorized convolutions which factorize a standard convolution into a depthwise convolution and a 1×1 convolution called a pointwise convolution.

For MobileNets the depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1×1 convolution to combine the outputs the depthwise convolution. A standard convolution both filters and combines inputs into a new set of outputs in one step. The depthwise separable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining. This factorization has the effect of drastically reducing computation and model size. First it uses depthwise separable convolutions to break the interaction between the number of output channels and the size of the kernel.

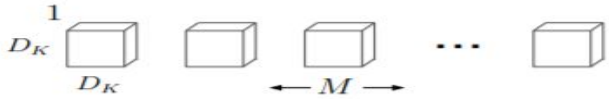
Depthwise separable convolution are made up of two layers: depthwise convolutions and pointwise convolutions. We use depthwise convolutions to apply a single filter per each input channel (input depth). Pointwise convolution, a simple 1×1 convolution, is then used to create a linear combination of the output of the depthwise layer. MobileNets use both batch-norm and ReLU nonlinearities for both layers.

Depthwise convolution is extremely efficient relative to standard convolution. However it only filters input channels, it does not combine them to create new features. So an additional layer that computes a linear combination of the output of depth wise convolution via 1 × 1 convolution is

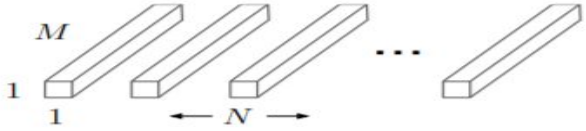
needed in order to generate these new features. The combination of depth wise convolution and 1 × 1 (pointwise) convolution is called depthwise separable convolution.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2. The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

Hyper-parameter 1:

Although the base MobileNet architecture is already small and low latency, many times a specific use case or application may require the model to be smaller and faster. In order to construct these smaller and less computationally expensive models we introduce a very simple parameter α called width multiplier. The role of the width multiplier α is to thin a network uniformly at each layer.

Width multiplier has the effect of reducing computational cost and the number of parameters quadratically by roughly α 2. Width multiplier can be applied to any model structure to define a new smaller model with a reasonable accuracy, latency and size trade off. It is used to define a new reduced structure that needs to be trained from scratch.

Hyper-parameter 2:

The second hyper-parameter to reduce the computational cost of a neural network is a resolution multiplier ρ. We apply this to the input image and the internal representation of every layer is

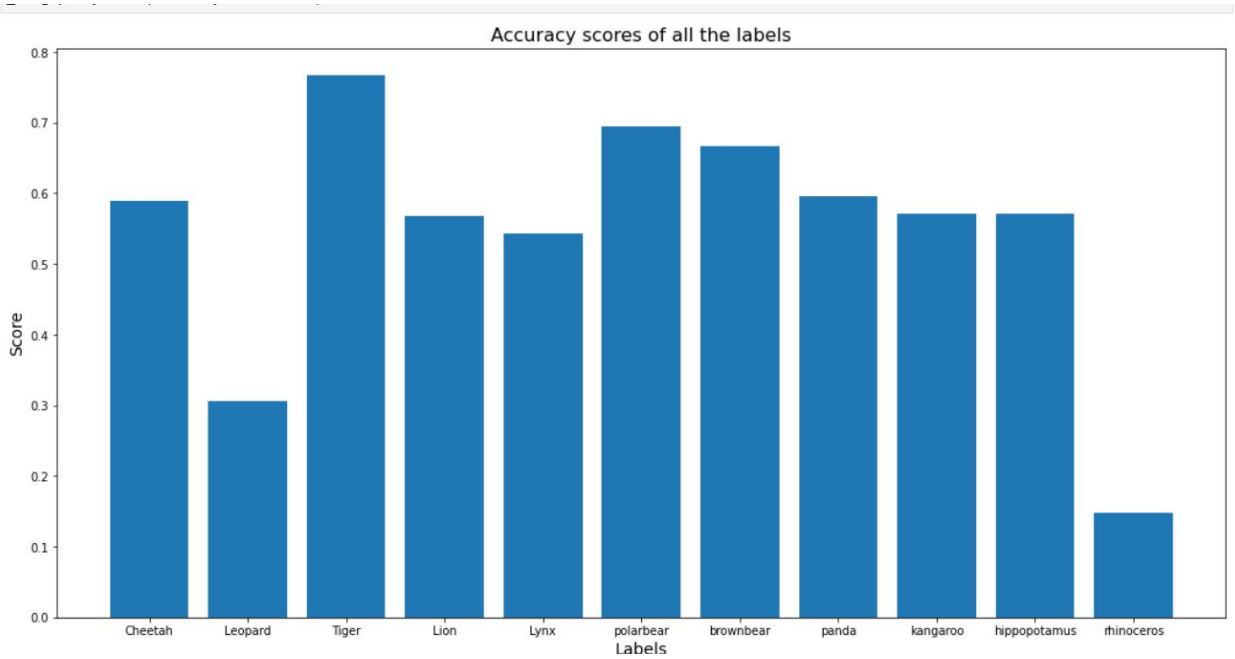
subsequently reduced by the same multiplier. In practice we implicitly set ρ by setting the input resolution.

MobileNets are small, low-latency, low-power models parameterized to meet the resource constraints of a variety of use cases. They effectively maximize accuracy while being mindful of the restricted resources for an on-device or embedded application.

But these advantages always come up with the disadvantages with MobileNet. Though it has reduced size, reduced parameters and performs slightly faster, it is slightly less accurate than the other models. When we searched for the image classifications done by other authors using MobileNet, ResNet50, VGG16 and InceptionResNetV2. The accuracies were nearly the same where InceptionResNet50 had the highest accuracy and MobileNet the least. And this difference was approximately ~5% between the highest and the least accurate model. Given the size of the network and the parameters to train, MobileNet performed a great job at classifying images with a little compromise in accuracy. Since we are working with large subsets of data, we wanted our computations to be faster with a little compromise in the accuracy of the model. Hence we took the MobileNet as our pre-trained model.

Accuracy of the model:

We trained our model on 11 set labels from the google image dataset. There was considerable variance in the accuracies of the predicted labels.



We can see that the model struggled to identify the images with 'Rhinoceros' and 'Leopard'. This could be due to the fact that the model has been insufficiently trained with the images of 'Rhinoceros' and 'Leopard'.

Ranking of Images:

Images are ranked based on the quality of the image and its resolution. Features of a picture can also have an effect in ranking images such as number of pixels, aspect ratio, image file size, image entropy and image gradient.

In the process of training image data using neural nets we are supposed to resize all the images to one input size of model before feeding image data to model. In most of the real time applications, image data is raw and has high variance in image features. Resizing such divergent data would condense or increase size of image which leads to poor image clarity with highly pixelated images or blurry images. Training models on such unrealistic images would result in bad performance of the model and leads to wrong predictions.

To avoid this problem and build our model with high predicting accuracy, we wanted to train the model with images having different labels and low variance in image features. As image size is

an important feature to avoid this problem. We have ranked images based on Pixels per Inch value computed for each image. Pixels per Inch is computed by using image dimensional features like Diagonal length of the screen, Width and Height.

$$Pixels per Inch = \sqrt{(width^2 + height^2)/DiagonalLength}$$

Using the above equation on each of the images and ranking in descending order based on the value generated by PPI would batch images with large size as least ranked and small size as highest ranked. Based on our interest to work with high resolution images or low resolution images we can pick images from the data using a certain range of rank.

Another way to solve this problem is using the PPI of model input size. As our input size of the model is 224x224, PPI for this size of image would be around 18 so taking images with PPI closer to 18 would also be ideal to build a better predicting model.

Though this was our plan of action before starting off, because of the computational incompetence of working on large amounts of data. We have made few changes during the application. As we are supposed to work with less amount of data and having minimum divergence is a necessity to build a better model, we didn't apply ranking on the raw data. We have proceeded to work with raw data generated with 11 different labels.

As the process of reverse image search our model has to extract features of random input image and search for a number of highly relevant image features from training data and print those images. To get print images with high resolution we have ranked images using ppi and print those images with a predicted label of random input image.

Reverse Image Search:

It is a content based image retrieval query technique that involves providing a CBIR (Content based information retrieval) system with a sample image as input that will then base its search upon; in terms of information retrieval and returns results related to the image. We use map-reduce for reverse image search. Images get mapped to features, features get mapped to new features and feature selection choses subsets of the required data.

Our ideology of applying reverse image search in this project is to detect objects in random sample input images as labels and search for those labels from trained data which has high PPI. Because of spark's lazy computation we were not able to perform these operations as a whole. We also tried matching features predicted on random input samples to that of trained data features and print those images with high relevance using k-means clustering but couldn't end up being successful because of memory overhead issues. We are only able to predict labels of random input images and search for high resolution images with that label.

As an input we have given a sample image as a URL. Then we defined a function to calculate the pixels per inch (ppi) of an image. Now we have defined a reverse image function using the model mobilenet and other prediction functions along with a parquet file for the model to read and first take the names of the labels from the original dataframe. Then taking names of the labels from predictions. Now we need to filter the images based on a subset of interest. Now, we can rank the images on pixel per inch (ppi). Finally images are mapped to features and the new features selection choses subsets of the required data and the images are displayed. Most of the images are drawings. Since we gave an image of a real lion and a tiger as a sample, we would expect the algorithm to predict a real image. There is a scope for improvement on this process. Nevertheless, it did predict the images with correct labels.

Results: We had given the image of a lion and a tiger and got the following results:

Input image:

Output: