Smartphone Identification and Cost Detection

Jake Garrison
6/3/2016
Department of Electrical Engineering
University of Washington
omonoid@uw.edu

Abstract

Convolutional neural networks have become the foundation for state of the art object detection and tracking tasks but often require a high-performance GPU to achieve near real-time results. Furthermore, many tasks require more than simply identifying and tracking an object and would benefit from the additional context that can be inferred from features in the image. We present a convolutional neural network trained to identify various smartphone models, as well as damaged phones, in order to estimate a dollar value for the phone. This network is optimized for low power, mobile usage and achieves near real-time predictions of live camera frames

1 Introduction

1.1 Contributions

Most recent advances in object detection and tracking utilize a convolutional neural network (CNN) to extract and learn features from pre-labeled images while training so that when presented with unseen images, can best choose a label from memory based on the features present. The feature extraction in a CNN is inspired by the human brain's visual cortex which breaks images into features such as edges, shapes, and corners, which then activates certain knowledge and memories that tie attributes such as name, quality, cost, size to the objects present in the image [1]. Most CNN research focuses on mapping objects in an image to a single attribute, often a name or label for the object. In this paper, we explore a method to obtain additional context about detected objects, specifically a quality attribute that can then be used to estimate the cost of the object.

An additional purpose of this paper is to demonstrate a CNN optimized for mobile use. We briefly show how a high-performance CNN model can be compressed and optimized for iOS use on a continuous stream of images with near real-time predictions. Many of the impressive applications enabled by CNNs are only useful if they can be executed on low power, portable hardware such as smartphones.

A final contribution lies in the methods designed and used to create a large dataset from existing images found in popular search engines.

While the topics discussed in this paper can be applied to any object with commercial value, smartphone identification task was chosen for the following reasons:

1. Different smartphone models often have unique features that distinguish it from other models, such as the camera placement, speaker cutout, or logo.
2. Most people have smartphones, so they have adequate familiarity to recognize differences between various models, as well as the usefulness of this technology.
3. There is a large market for smartphones which means there are several images on the internet, making it easy to build a comprehensive image database of various phone models.

1.2 Related Work

Although research focused on using computers to simulate brain functions has existed since the 1950's, an early working example of a CNN was published in 1998 by Yann LeCun, known as LeNet-5 which classified written digits [2]. Since then, GPU's have come into play allowing CNNs to execute faster and with more layers of complexity. In 2012, CNNs came back into the spotlight with a publication that discussed the first 'deep' CNN, titled AlexNet, which shattered prior records in the ImageNet Large-Scale Visual Recognition Challenge [3]. This prompted Microsoft, Google, and Facebook, to name a few to continually improve the capabilities of CNNs every year.

While there doesn't appear to be any prior work on identifying phone models or estimating phone value, there are several examples involving the use of CNNs for object detection. These applications range from medical to astrophysics to autonomous driving.

This paper is based on Google's Inception model [4] which came out in 2015 and introduced several new ideas to the standard and most popular CNN architecture demonstrated in AlexNet. Google's Tensorflow framework [5] is used to implement the Inception architecture, as well as to train and execute it on both a workstation and mobile phone. More details on this are presented in the Phone Classifier section.

Others have created a dataset from internet images, but few have documented the process. The majority of prior research is about unsupervised techniques [6]. There is no mention of utilizing pretrained CNNs for dataset sorting (see Dataset Filtering section).

2 Dataset

A dataset of labeled phone models is crucial for creating a robust classifier. Since no datasets exist, one had to be built from scratch. The automated dataset construction can be broken into three steps, image aggregation, filtering, and clustering.

2.1 Aggregation

As mentioned, smartphones were selected for this project due to the availability of images on the internet. Many people upload images of their smartphone when they attempt to sell them, and manufacturers often post several stock images online to introduce new models or upgrades. Image databases such as Google Images or Bing cluster these images so they can be queried using search keywords. The automated filing done be these services is taken advantage of to build a local dataset of phone images indexed by their make and model.

For this project, an image scraper built in Node.js used existing libraries for querying [7] and downloading [8] images from popular image provider services (Google, Yahoo, Bing, and Baidu). Phone makes and models can be queried along with additional keywords such as 'for sale', 'new' or 'broken' and a field for how many images to retrieve. Using this tool and a list of keywords and models to query, a large dataset of over 30 thousand images was created. Unfortunately, this step relied on the search engine's ability to correctly index images and subsequently images of ads or accessories for the phone were frequently downloaded as false positives, or the images were duplicates or corrupted. Further filtering is necessary to ensure all images contain a phone.

2.2 Filtering

A filtering program was developed to fine tune the image dataset to omit corrupt, similar/duplicate images and images that don't contain a phone. For detecting corrupt images, the MIME type [9] was extracted from each image and checked to ensure the data in the file was indeed an image. In addition, all images were converted to .jpg. Any file that was not considered an image was removed.

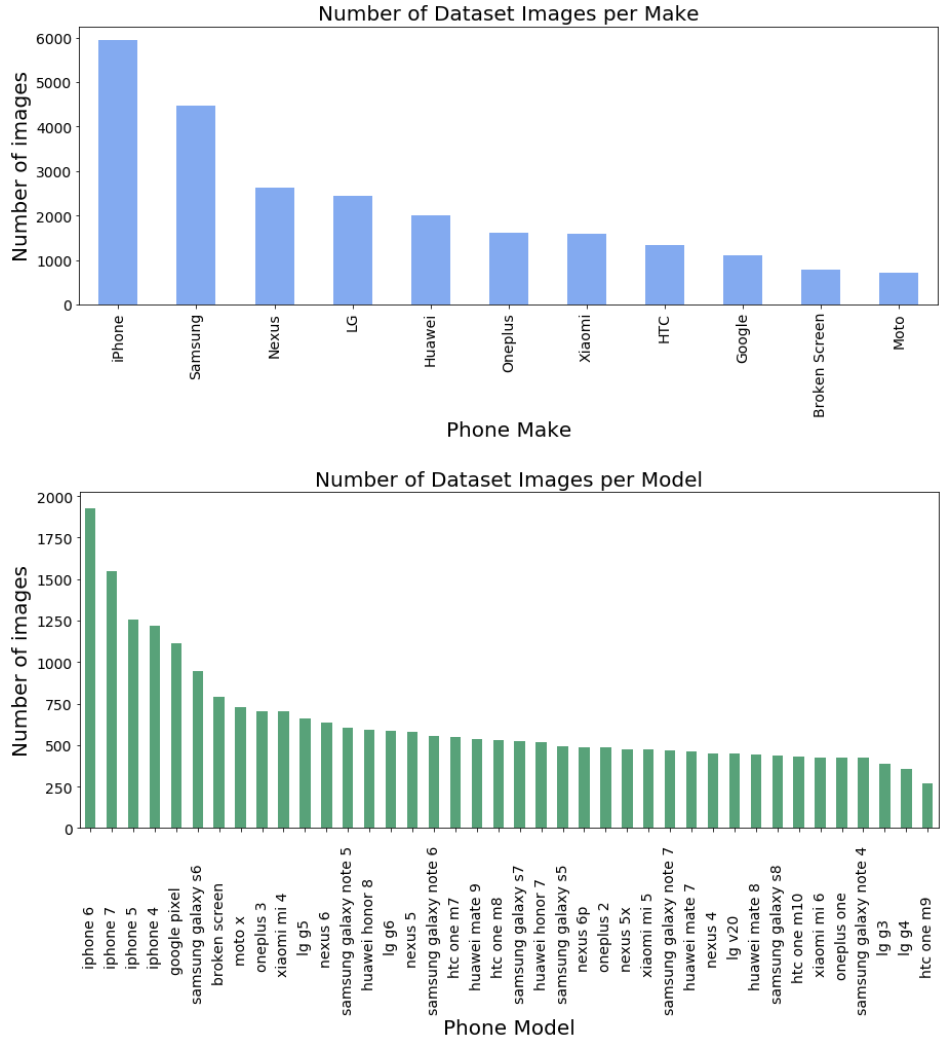
Next, the OpenCV Scale-invariant feature transform (SIFT) implantation [10] was used to find highly similar or duplicate images. It was often the case that various sizes of the same image were scattered throughout the dataset, this step removed copies that were similar enough.

Finally, to identify images not actually containing a phone, a CNN [11] pretrained to identify common objects such as TV's, people and phones [11] was utilized. This CNN extracts and labels a

bounding box for any object it recognizes. If an image didn't contain a 'cell phone', it is flagged for removal, after this step over 5 thousand images were identified as false positives and removed.

2.3 Clustering

With most false positives removed, the images could be clustered into different labels. For this application, the specific labels were initially decided to be the model of the phone such as iPhone 5s; however, it was later determined that further grouping was necessary as phone models such as iPhone 5, 5s and 5s Plus did not have adequate distinguishing features to differentiate them. As a result, labels were subjectively clustered together based on physical similarity. An additional 'broken phone' label was added to help the CNN identify qualities independent of the make and model that indicate loss of quality, such as shattered screen or dented corners. Finally, all labels were named with the following formula {"Phone make" + "model"}, such as "iPhone" + "5s" or "Samsung" + "Galaxy S6" so the make and model could be easily extracted from the label. Eleven different phone makes, comprised of 40 unique models, were declared as a result of clustering. See Figures 1a and 1b for the make and model distributions.



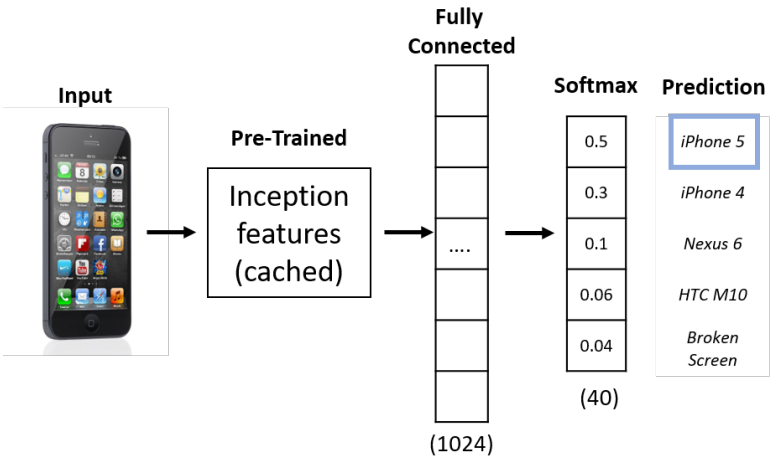
Figures 1a and 1b: Show the image count distribution for phone makes and models respectively

Ideally, the histograms are roughly uniform indicated an equal number of images for each label, but

due to the clustering explained above and the availability of images, the labels are more imbalanced. It was found in the training phase that this didn't cause the CNN to have an unfair bias toward the more popular labels. In addition, the above distribution generally represents the popularity of the phone model in America.

3 Phone Classifier

The Inception model (v3), powered by Tensorflow [4], was used as the CNN architecture for identifying phone models. Rather than retraining the several layers of inception which can take weeks, transfer learning was used to retrain the final layer of the model, the fully connected layer, which is responsible for turning the extracted features into a prediction. More information on transfer learning can be found in the original paper [13]. While it was not as good as training all layers from scratch, it performed surprisingly well and only took an hour rather than several days allowing for extra time to be spent on optimizing the training process. See Figure 2 below for a diagram outlining the transfer learning process.



Figures 2: Outlines the transfer learning process

The input images were passed through several cached Inception layers that were pretrained to extract features such as edges, corners, and other relationships. The final extracted features were represented by a single fully connected vector of length 1024. This was then compressed into the softmax vector which had a length equal to the number of labels (40). Each entry of this vector was a confidence or likelihood number between 0 and 1, which corresponds to the certainty of that entry being the final prediction. The entry with the highest confidence was the networks top guess for the predicted label. Note all confidences were normalized to sum to one.

3.1 Training

A training routine was written to load the pretrained inception weights and retrain the last layer with the custom phone dataset. Random brightness and flipping was applied to the training images to expand the size of the dataset and train the model to be robust to different lighting and rotations of the phones. The dataset was randomly split into a train and test set. During training, the train and validation accuracy were reported periodically, and the final test accuracy was evaluated after training concludes. The final hyperparameters and learning curve is showed below in Figure 3.

Hyperparameters:

Epochs: *80,000*
Learning Rate: *0.01*
Batch size: *100*
Train/Val/Test Split: *80/10/10*
Random brightness
Random flipping

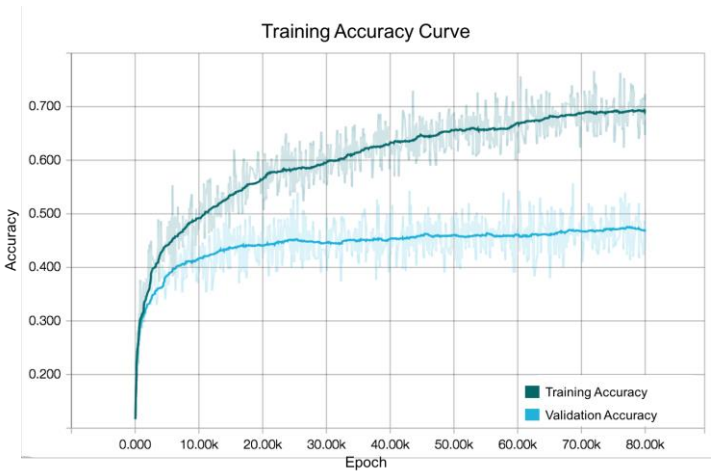


Figure 3: Training parameters and learning curve

The network was trained to minimize the cross-entropy loss. After 30 thousand epochs, the validation flattens indicating learning is slowing down. As more epochs were completed, the validation curve approached a horizontal asymptote and was no longer able to learn anything new from the dataset. For this reason, only 80 thousand epochs were completed in training.

3.2 Evaluation

The network's accuracy was evaluated in two ways. First, the model prediction accuracy was measured as the percentage of images in which the phone model was correctly identified. It was found that the correct make of the phone was accurately predicted; however, the model of the phone was periodically incorrect in some cases; for example, an iPhone 6 being predicted as an iPhone 7. Even though this was an error, the inaccuracy was not as significant as incorrectly predicting an iPhone 6 as a Samsung S6. This observation, which can be seen visually in the confusion matrix in Figure 4a, led to a second evaluation metric, the phone make accuracy. The make accuracy is simply how often the extracted make of the final prediction is correct. Both evaluation metrics, along with the final training and validation accuracies are displayed in Table 1 below.

Training	Validation	Test (Make)	Test (Model)
75%	51%	73%	54%

Table 1: Evaluation results

In addition to evaluating the prediction accuracy, the softmax layer's confidence entries were also analyzed. Ideally, there was a single phone model with much higher confidence than the others, but it is often the case that the predicted model has confidence less than 0.4, which given that there are 40 models, is still often dominant. The box plots in Figure 4b below compare the confidence in the top choice for both the make and model attribute. As expected, the make confidence is generally much higher.

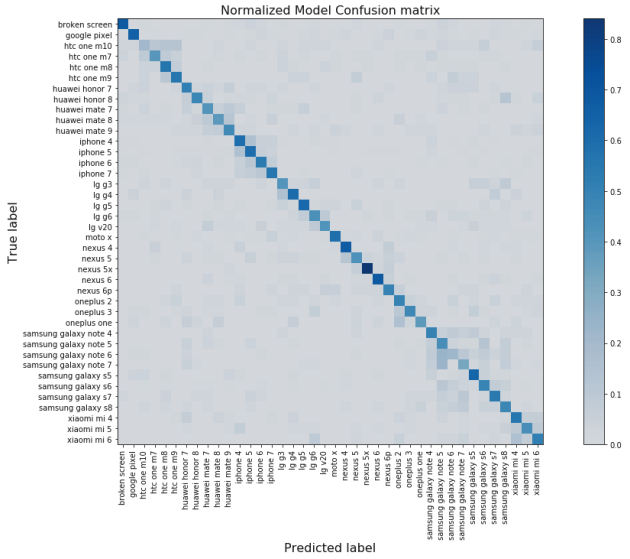


Figure 4a: Confusion matrix indicating the phone model accuracy of the test set. The gray clumps near the diagonals highlight the correct make but incorrect model predictions.

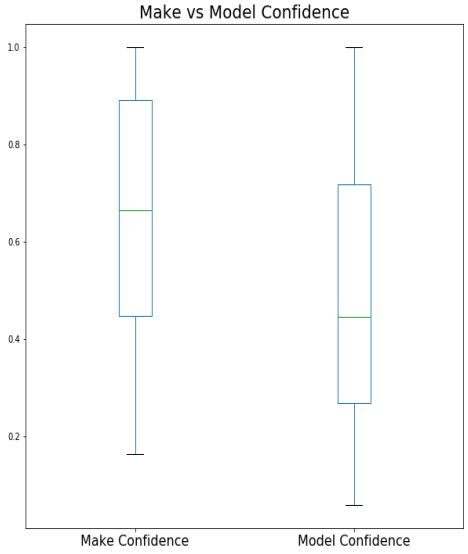


Figure 4b: Box plots showing the confidence distribution of the two evaluation metrics

4 Cost Estimation

Up to this point, the network can take an input image and predict the phone make and model, which as mentioned in the introduction, is not entirely groundbreaking or useful. One of the added contributions of this paper is to introduce further context that can be inferred from the image, in this case, the estimated value or cost of a given phone.

Cost is calculated as a weighted sum of the market price of each phone, weighted by the confidence outputted by the network's softmax layer. Next, a discount is applied based on the confidence level of the 'broken phone' label. This way phones with a greater degree of wear and tear or broken screens are devalued. The equation below summarizes this calculation.

$$estimated\ cost = discount \times \sum_i confidence_i \times cost_i$$

For example, if the network outputs 0.5, 0.3, 0.10 and 0.06 (as depicted in Figure 2 above) for four phone models that cost of $120, $180, $130, and $210 respectively, and the broken label has a confidence of 0.04 or discount of 4%. The equation to the right shows how the final output cost is calculated.

$$(1 - 0.04)\left(\begin{bmatrix}0.50\\0.30\\0.10\\0.06\end{bmatrix} \cdot \begin{bmatrix}\$120\\\$180\\\$130\\\$210\end{bmatrix}\right) = \$134$$

The market price is based on live database listings [14] which assign a cost value based on recent new or used sales of a given item. These costs are stored in a local database for each phone model and are shown in Figure 5 below.

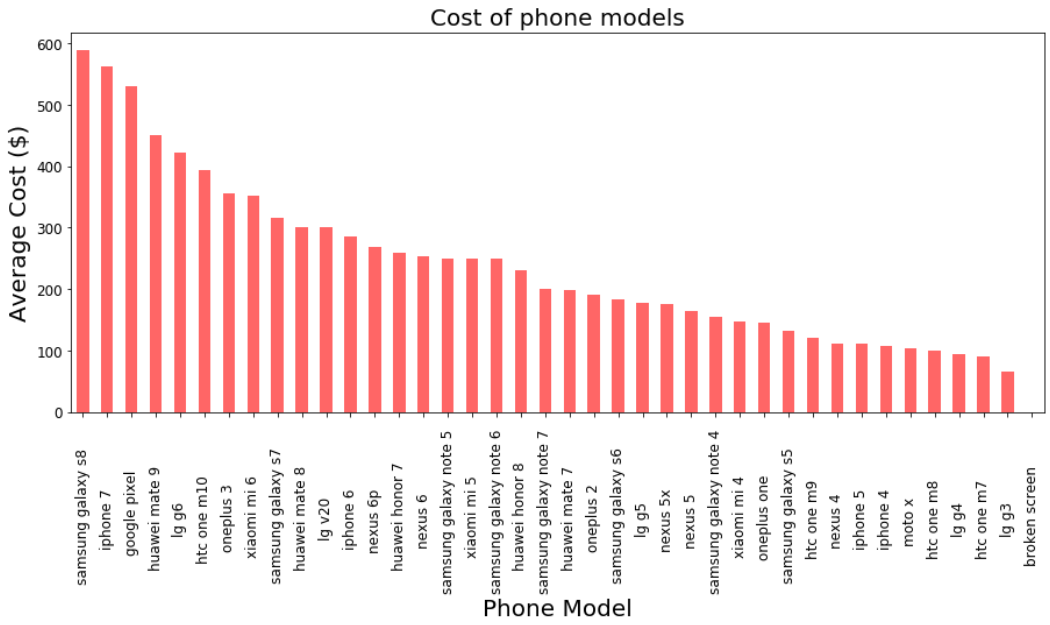


Figure 5: Distribution of phone cost for each model

5 iOS App

Using select tools provided in the Tensorflow framework, a simple iOS demo app was built to demonstrate the trained CNN and cost algorithm in a real-time scenario.

5.1 Optimizations

Another contribution of this paper is to demonstrate a working, useful CNN on low-powered, portable hardware. To do this, the CNN first needed to be optimized for iOS execution. A script with three steps was written to achieve this:

1. The input of the CNN was changed from a jpg image to a raw iOS camera buffer. This way consecutive image frames could be piped through the network without needing conversion to jpg.
2. The CNN's floating point weights were quantized to have less precision, which ultimately makes the computations faster and the cached weights occupy less space.
3. The model file was mapped to iOS memory so it loaded on demand.

With these optimizations, the model loaded and ran much faster at the cost of slightly lower accuracy due to the quantized weights.

5.2 User Interface

When the app is open, camera frames are instantly and sequentially evaluated by the network. Softmax entries with a confidence above 5% are displayed in descending order. The estimated cost output is also displayed above the predictions. In order to smooth the output displayed, a running sum parameterized by an update and decay value is used. This gives the app a very crude form of memory which allows it to converge to a prediction as more frames are processed. If no phone is seen (the confidences are all below 0.05), a message is displayed in place of the predictions and cost.

A circle button to pause the app and take a snapshot is also included so the user can save the prediction in the form of a screenshot. There is also a camera swap button that toggles between the front and back camera for input. Screenshots of predictions for several different phones are shown in Figure 6 below. Note that the confidence is generally higher for the backside of the phones since there are unique features on the backside of phones (most screens on the front look similar for all phone models). Also, note that the top right phone in Figure 6 has a broken screen which is correctly

identified and valued at a much lower rate.

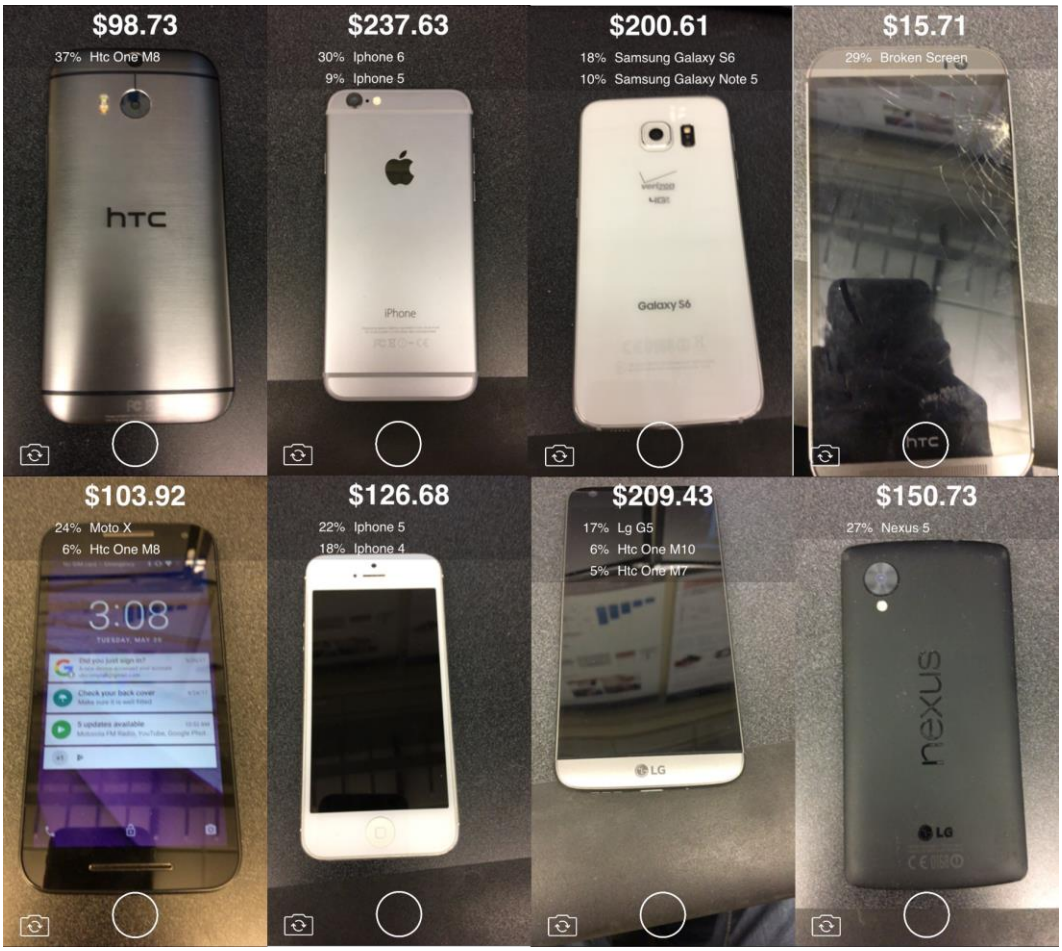


Figure 6: Prediction screenshots from iOS app

6 Future Work

To conclude, this paper highlighted three contributions, each of which has plenty of room for improvement:

1. The steps to automate dataset construction using images from popular search engines,
2. The use of a CNN to estimate multiple attributes from an image, in this case, phone make, model and estimated cost.
3. How a CNN can be optimized for real-time execution on a low powered portable device.

In the dataset creation steps, search query tricks can be used to increase the likelihood of capturing realistic images of phones rather than renderings or ads. In addition, unsupervised clustering can be used to isolate images that don't look like phones. Finally, using Craigslist or eBay to obtain both images and a sale price would allow the network to better understand features that devalue a given phone model.

In the training phase, a custom CNN architecture could be built and trained from scratch specifically for phones, rather than using transfer learning. Also, more distortion techniques such as random cropping or scaling can be used to help the model generalize and train faster. A custom cost function could replace the cross-entropy metric so predictions that are the correct make, but the wrong model are not penalized as much.

There are several improvements that could benefit the iOS app. The smoothing of the displayed predictions has a lot of room for improvement. For example, a Long short-term memory (LSTM) [15] could be used to handle the predictions for the incoming sequences of frames. This would add memory cells so the predictions displayed can be a learned function of the previous frames, rather than a linear combination. Rather than displaying an endless stream of predictions, the app could stop predicting once the change in prediction values converges. This way the app would stop 'thinking' once it is has reached a high enough level of certainty.

References

[1] "Convolutional Neural Networks (LeNet)." Convolutional Neural Networks (LeNet) — DeepLearning 0.1 documentation. Accessed June 04, 2017. http://deeplearning.net/tutorial/lenet.html.

[2] LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86, no. 11 (1998): 2278-2324.

[3] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097-1105. 2012.

[4] Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-9. 2015.

[5] Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." *arXiv preprint arXiv:1603.04467* (2016).

[6] Rubinstein, Michael, Armand Joulin, Johannes Kopf, and Ce Liu. "Unsupervised joint object discovery and segmentation in internet images." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1939-1946. 2013.

[7] Pevers. "Images-scraper." Npm. Accessed June 04, 2017. https://www.npmjs.com/package/images-scraper.

[8] Demsking. "Image-downloader." Npm. Accessed June 04, 2017. https://www.npmjs.com/package/image-downloader.

[9] "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies." IETF Tools. Accessed June 04, 2017. https://tools.ietf.org/html/rfc2045.

[10] Lowe, Toy building set. U.S. Patent 6711293 B1 filed Mar 6, 2000, and issued Mar 23, 2004.

[11] Lin, Tsung-Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. "Microsoft coco: Common objects in context." In *European Conference on Computer Vision*, pp. 740-755. Springer International Publishing, 2014.

[12] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[13] Donahue, Jeff, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition." In *Icml*, vol. 32, pp. 647-655. 2014.

[14] "What is the market price for..." Find out the market price for anything - The Price Geek. Accessed June 04, 2017. http://www.thepricegeek.com/.

[15] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9, no. 8 (1997): 1735-1780.